Optimization for deep learning: theory and algorithms

Ruoyu Sun *

December 21, 2019

Abstract

When and why can a neural network be successfully trained? This article provides an overview of optimization algorithms and theory for training neural networks. First, we discuss the issue of gradient explosion/vanishing and the more general issue of undesirable spectrum, and then discuss practical solutions including careful initialization and normalization methods. Second, we review generic optimization methods used in training neural networks, such as SGD, adaptive gradient methods and distributed methods, and existing theoretical results for these algorithms. Third, we review existing research on the global issues of neural network training, including results on bad local minima, mode connectivity, lottery ticket hypothesis and infinite-width analysis.

1 Introduction

A major theme of this article is to understand the practical components for successfully training neural networks, and the possible factors that cause the failure of training. Imagine you were in year 1980 trying to solve an image classification problem using neural networks. If you wanted to train a neural network from scratch, it is very likely that your first few attempts would have failed to return reasonable results. What are the essential changes to make the algorithm work? In a high-level, you need three things (besides powerful hardware): a proper neural network, a proper training algorithm, and proper training tricks.

- Proper neural-net. This includes neural architecture and activation functions. For neural architecture, you may want to replace a fully connected network by a convolutional network with at least 5 layers and enough neurons. For better performance, you may want to increase the depth to 20 or even 100, and add skip connections. For activation functions, a good starting point is ReLU activation, but using tanh or swish activation is also reasonable.

- Training algorithm. A big choice is to use stochastic versions of gradient descent (SGD) and stick to it. A well-tuned constant step-size is good enough, while momentum and adaptive stepsize can provide extra benefits.

*Department of Industrial and Enterprise Systems Engineering (ISE), and affiliated to Coordinated Science Laboratory and Department of ECE, University of Illinois at Urbana-Champaign, Urbana, IL. Email: ruoyus@illinois.edu.

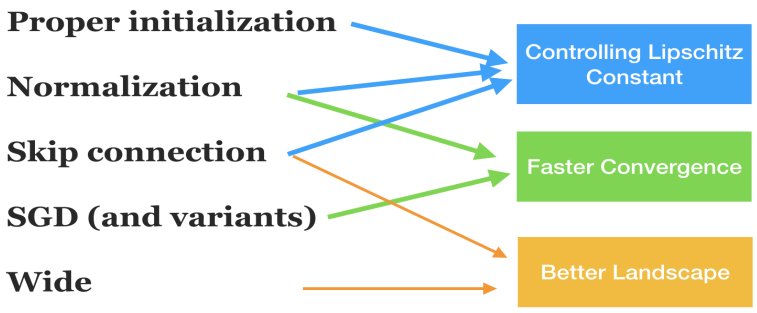


Figure 1: A few major design choices for a successful training of a neural network with theoretical understanding. They have impact on three aspects of algorithm convergence: make convergence possible, faster convergence and better global solutions. The three aspects are somewhat related, and it is jut a rough classification. Note that there are other important design choices, especially the neural architecture, that is not understood theoretically, and thus omitted in this figure. There are also other benefits such as generalization, which is also omitted.

- Training tricks. Proper initialization is very important for the algorithm to start training. To train a network with more than 10 layers, two extra tricks are often needed: adding normalization layers and adding skip connections.

Which of these design choices are essential? Currently we have some understanding of a few design choices, including initialization strategies, normalization methods, the skip connections, over-parameterization (large width) and SGD, as shown in Figure 1. We roughly divide the optimization advantage into three parts: controlling Lipschitz constants, faster convergence and better landscape. There are many other design choices that are hard to understand, most notably the neural architecture. Anyhow, it seems impossible to understand every part of this complicated system, and the current understanding can already provide some useful insight.

To keep the survey simple, we will focus on the supervised learning problem with feedforward neural networks. We will not discuss more complicated formulations such as GANs (generative adversarial networks) and deep reinforcement learning, and do not discuss more complicated architecture such as RNN (recurrent neural network), attention and Capsule. In a broader context, theory for supervised learning contains at least representation, optimization and generalization (see Section 1.1), and we do not discuss representation and generalization in detail. One major goal is to understanding how the *neural-net structure* (the parameterization by concatenation of many variables) affects the design and analysis of optimization algorithms, which can potentially go beyond supervised learning.

This article is written for researchers who are interested in theoretical understanding of optimization for neural networks. Prior knowledge on optimization methods and basic theory will be very helpful (see, e.g., [24, 200, 30] for preparation). Existing surveys on optimization for deep learning are intended for general machine learning audience, such as Chapter 8 of the book Goodfellow et al. [77]. These reviews often do not discuss optimization theoretical aspects in depth. In contrast, in this article, we emphasize more on the theoretical results while trying to make it accessible for non-theory readers. Simple examples that illustrate the intuition will be provided if possible, and we will not explain the details of the theorems.

1.1 Big picture: decomposition of theory

A useful and popular meta-method to develop theory is decomposition. We first briefly review the role of optimization in machine learning, and then discuss how to decompose the theory of optimization for deep learning.

Representation, optimization and generalization. The goal of supervised learning is to find a function that approximates the underlying function based on observed samples. The first step is to find a rich family of functions (such as neural networks) that can represent the desirable function. The second step is to identify the parameter of the function by minimizing a certain loss function. The third step is to use the function found in the second step to make predictions on unseen test data, and the resulting error is called test error. The test error can be decomposed into representation error, optimization error and generalization error, corresponding to the error caused by each of the three steps.

In machine learning, the three subjects representation, optimization and generalization are often studied separately. For instance, when studying representation power of a certain family of functions, we often do not care whether the optimization problem can be solved well. When studying the generalization error, we often assume that the global optima have been found (see [96] for a survey of generalization). Similarly, when studying optimization properties, we often do not explicitly consider the generalization error (but sometimes we assume the representation error is zero).

Decomposition of optimization issues. Optimization issues of deep learning are rather complicated, and further decomposition is needed. The development of optimization can be divided into three steps. The first step is to make the algorithm start running and converge to a reasonable solution such as a stationary point. The second step is to make the algorithm converge as fast as possible. The third step is to ensure the algorithm converge to a solution with a low objective value (e.g. global minima). There is an extra step of achieving good test accuracy, but this is beyond the scope of optimization. In short, we divide the optimization issues into three parts: convergence, convergence speed and global quality.

$$\text{Optimization issues} \begin{cases} \text{Local issues} \begin{cases} \text{Convergence issue: gradient explosion/vanishing} \\ \text{Convergence speed issue} \end{cases} \\ \text{Global issues: bad local minima, plaeaus, etc.} \end{cases}$$

Most works are reviewed in three sections: Section 4, Section 5 and Section 6. Roughly speaking, each section is mainly motivated by one of the three parts of optimization theory. However, this partition is not precise as the boundaries between the three parts are blurred. For instance, some techniques discussed in Section 4 can also improve the convergence rate, and some results in Section 6 address the convergence issue as well as global issues. Another reason of the partition is that they represent three rather separate subareas of neural network optimization, and are developed somewhat independently.

1.2 Outline

The structure of the article is as follows. In Section 2, we present the formulation of a typical neural network optimization problem for supervised learning. In Section 3, we present back propagation (BP) and analyze the difficulty of applying classical convergence analysis to gradient descent for neural networks. In Section 4, we discuss neural-net specific tricks for training a neural network, and some underlying theory. These are neural-network dependent methods, that open the black box of neural networks. In particular, we discuss a major challenge called gradient explosion/vanishing and a more general challenge of controlling spectrum, and review main solutions such as careful initialization and normalization methods. In Section 5, we discuss generic algorithm design which treats neural networks as generic non-convex optimization problems. In particular, we review SGD with various learning rate schedules, adaptive gradient methods, large-scale distributed training, second order methods and the existing convergence and iteration complexity results. In Section 6, we review research on global optimization of neural networks, including global landscape, mode connectivity, lottery ticket hypothesis and infinite-width analysis (e.g. neural tangent kernel).

2 Problem Formulation

In this section, we present the optimization formulation for a supervised learning problem. Suppose we are given data points $x_i \in \mathbb{R}^{d_x}, y_i \in \mathbb{R}^{d_y}, i = 1, \ldots, n$, where n is the number of samples. The input instance x_i can represent a feature vector of an object, an image, a vector that presents a word, etc. The output instance y_i can represent a real-valued vector or scalar such as in a regression problem, or an integer-valued vector or scalar such as in a classification problem.

We want the computer to predict y_i based on the information of x_i, so we want to learn the underlying mapping that maps each x_i to y_i. To approximate the mapping, we use a neural network $f_\theta : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$, which maps an input x to a predicted output $\hat{y}$. A standard fully-connected neural network is given by

$$f_\theta(x) = W^L \phi(W^{L-1} \ldots \phi(W^2 \phi(W^1 x))), \tag{1}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the neuron activation function (sometimes simply called "activation" or "neuron"), W^j is a matrix of dimension $d_j \times d_{j-1}$, $j = 1, \ldots, L$ and $\theta = (W^1, \ldots, W^L)$ represents the collection of all parameters. Here we define $d_0 = d_x$ and $d_L = d_y$. When applying the scalar function ϕ to a matrix Z, we apply ϕ to each entry of Z. Another way to write down the neural network is to use a recursion formula:

$$z^0 = x; \quad z^l = \phi(W^l z^{l-1}),\ l = 1, \ldots, L. \tag{2}$$

Note that in practice, the recursive expression should be $z^l = \phi(W^l z^{l-1} + b^l)$. For simplicity of presentation, throughout the paper, we often skip the "bias" term b^l in the expression of neural networks and just use the simplified version (2).

We want to pick the parameter of the neural network so that the predicted output $\hat{y}_i = f_\theta(x_i)$ is close to the true output y_i, thus we want to minimize the distance between y_i and $\hat{y}_i$. For a

certain distance metric $\ell(\cdot,\cdot)$, the problem of finding the optimal parameters can be written as

$$\min_{\theta} F(\theta) \triangleq \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i)). \tag{3}$$

For regression problems, $\ell(y,z)$ is often chosen to be the quadratic loss function $\ell(y,z) = \|y-z\|^2$. For binary classification problem, a popular choice of ℓ is $\ell(y,z) = \log(1+\exp(-yz))$.

Technically, the neural network given by (2) should be called fully connected feed-forward networks (FCN). Neural networks used in practice often have more complicated structure. For computer vision tasks, convolutional neural networks (CNN) are standard. In natural language processing, extra layers such as "attention" are commonly added. Nevertheless, for our purpose of understanding the optimization problem, we mainly discuss the FCN model (2) throughout this article, though in few cases the results for CNN will be mentioned.

For a better understanding of the problem (20), we relate it to several classical optimization problems.

2.1 Relation with Least Squares

One special form of (20) is the linear regression problem (least squares):

$$\min_{w\in\mathbb{R}^{d\times 1}} \|y - w^T X\|^2, \tag{4}$$

where $X = (x_1, \ldots, x_n) \in \mathbb{R}^{d\times n}$, $y \in \mathbb{R}^{1\times n}$. If there is only one linear neuron that maps the input x to $w^T x$ and the loss function is quadratic, then the general neural network problem (20) reduces to the least square problem (4). We explicitly mention the least square problem for two reasons. First, it is one of the simplest forms of a neural network problem. Second, when understanding neural network optimization, researchers have constantly resorted to insight gained from analyzing linear regression.

2.2 Relation with Matrix Factorization

Neural network optimization (20) is closely related to a fundamental problem in numerical computation: matrix factorization. If there is only one hidden layer of linear neurons and the loss function is quadratic, and the input data matrix X is the identity matrix, the problem (20) reduces to

$$\min_{W_1,W_2} \|Y - W_2 W_1\|_F^2, \tag{5}$$

where $W_2 \in \mathbb{R}^{d_y\times d_1}$, $W_1 \in \mathbb{R}^{d_1\times n}$, $Y \in \mathbb{R}^{d_y\times n}$ and $\|\cdot\|_F$ indicates the Frobenious norm of a matrix. If $d_1 < \min\{n, d_y\}$, then the above problem gives the best rank-d_1 approximation of the matrix Y. Matrix factorization is widely used in engineering, and it has many popular extensions such as non-negative matrix factorization and low-rank matrix completion. Neural network can be viewed as an extension of two-factor matrix factorization to multi-factor nonlinear matrix factorization.

3 Gradient Descent: Implementation and Basic Analysis

A large class of methods for neural network optimization are based on gradient descent (GD). The basic form of GD is

$$\theta_{t+1} = \theta_t - \eta_t \nabla F(\theta_t), \tag{6}$$

where η_t is the step-size (a.k.a. "learning rate") and $\nabla F(\theta_t)$ is the gradient of the loss function for the t-th iterate. A more practical variant is SGD (Stochastic Gradient Descent): at the t-th iteration, randomly pick i and update the parameter by

$$\theta_{t+1} = \theta_t - \eta_t \nabla F_i(\theta_t),$$

where $F_i(\theta) \triangleq \ell(y_i, f_\theta(x_i))$. We will discuss SGD in more detail in Section 5; in this section we will only consider simple GD and SGD.

In the rest of the section, we first discuss the computation of the gradient by "backpropagation", then discuss classical convergence analysis for GD.

3.1 Computation of Gradient: Backpropagation

The discovery of backpropagation (BP) was considered an important landmark in the history of neural networks. From an optimization perspective, it is just an efficient implementation of gradient computation [1]. To illustrate how BP works, suppose the loss function is quadratic and consider the *per-sample* loss of the non-linear network problem $F_i(\theta) = \|y_i - W^L \phi(W^{L-1} \dots W^2 \phi(W^1 x_i))\|^2$. The derivation of BP applies to any i, thus for simplicity of presentation we ignore the subscript i, and use x and y instead. In addition, to distinguish the per-sample loss with the total loss $F(\theta)$, we use $F_0(\theta)$ to denote the per-sample loss function:

$$F_0(\theta) = \|y - W^L \phi(W^{L-1} \dots W^2 \phi(W^1 x))\|^2. \tag{7}$$

We define an important set of intermediate variables:

$$\begin{aligned} z^0 &= x, & h^1 &= W^1 z^0, \\ z^1 &= \phi(h^1), & h^2 &= W^2 z^1, \\ &\vdots, & &\vdots \\ z^{L-1} &= \phi(h^{L-1}), & h^L &= W^L z^{L-1}. \end{aligned} \tag{8}$$

Here, h^l is often called pre-activation since it is the value that flows into the neuron, and z^l is called post-activation since it is the value comes out of the neuron. Further, define $D^l = \mathrm{diag}(\phi'(h_1^l), \dots, \phi'(h_{d_l}^l))$, which is a diagonal matrix with the t-th diagonal entry being the derivative of the activation function evaluated at the t-th pre-activation h_t^l.

[1] While using GD to solve an optimization problem is straightforward, discovering BP is historically nontrivial.

Let the error vector $e = 2(h^L - y)$ [2]. The gradient over weight matrix W^l is given by

$$\frac{\partial F_0}{\partial W^l} = (W^L D^{L-1} \dots W^{l+2} D^{l+1} W^{l+1} D^l)^T e (z^{l-1})^T, \quad l = 1, \dots L. \tag{9}$$

Define a sequence of backpropagated error as

$$\begin{aligned} e^L &= e, \\ e^{L-1} &= (D^{L-1} W^L)^T e^L, \\ &\dots, \\ e^1 &= (D^1 W^2)^T e^2. \end{aligned} \tag{10}$$

Then the partial gradient can be written as

$$\frac{\partial F_0}{\partial W^l} = e^l (z^{l-1})^T, \quad l = 1, 2, \dots, L. \tag{11}$$

This expression does not specify the details of computation. A naive method to compute all partial gradients would require $O(L^2)$ matrix multiplications since each partial gradient requires $O(L)$ matrix multiplications. Many of these multiplication are repeated, thus a smarter algorithm is to reuse the multiplications, similar to the memorization trick in dynamical programming. More specifically, the algorithm back-propagation computes all partial gradients in a forward pass and a backward pass. In the forward pass, from the bottom layer 1 to the top layer L, post-activation z^l is computed recursively via (8) and stored for future use. After computing the last layer output $f_\theta(x) = h^L$, we compare it with the ground-truth y to obtain the error $e = \ell(h^L, y)$. In the backward pass, from the top layer L to the bottom layer 1, two quantities are computed at each layer l. First, the backpropagated error e^l is computed according to (10), i.e., left-multiplying e^{l+1} by the matrix $(D^{l-1} W^l)^T$. Second, the partial gradient over the l-th layer weight matrix W^l is computed by (11), i.e., multiply the backward signal e^l and the pre-stored feedforward signal $(z^{l-1})^T$. After the forward pass and the backward pass, we have computed the partial gradient for each weight (for one sample x).

By a small modification to this procedure, we can implement SGD as follows. In the backward pass, for each layer l, after computing the partial gradient over W^l, we update W^l by a gradient step. After updating all weights W^l, we have completed one iteration of SGD. In mini-batch SGD, the implementation is slightly different: in the feedforward and backward pass, a mini-batch of multiple samples will pass the network together.

Rigorously speaking, the term "backpropagation" refers to algorithm that computes the partial gradients, i.e., for a mini-batch of samples, computing the partial gradients in one forward pass and one backward pass. Nevertheless, it is also often used to describe the entire learning algorithm, especially SGD.

[2] If the loss function is not quadratic, but a general loss function $\ell(y, h^L)$, we only need to replace $e = 2(h^L - y)$ by $e = \frac{\partial \ell}{\partial h^L}$.

3.2 Basic Convergence Analysis of GD

In this subsection, we discuss what classical convergence results can be applied to a neural network problem with minimal assumptions. Convergence analysis tailored for neural networks under strong assumptions will be discussed in Section 6. Consider the following question:

$$\text{Does gradient descent converge for neural network optimization (20)?} \tag{12}$$

Meaning of "convergence". There are multiple criteria of convergence. Although we wish that the iterates converge to a global minimum, a more common statement in classical results is "every limit point is a stationary point" (e.g. [24]). Besides the gap between stationary points and global minima (will be discussed in Section 6), this claim does not exclude a few undesirable cases: (U1) the sequence could have more than one limit points; (U2) limit points could be non-existent [3], i.e., the sequence of iterates can diverge. Eliminating (U1) and (U2) to ensure convergence to a single stationary point is not easy; see Appendix A for more discussions.

Another criterion is the convergence of function values. This kind of convergence is very easy to achieve: if the function value is lower bounded by 0 and the sequence $F(\theta_t)$ is decreasing, then the sequence must converge to some finite value $\hat{F}$. However, optimizers do not regard this as a meaningful criterion since $\hat{F}$ could be an arbitrary value.

In this section, we focus on a meaningful and simple convergence criterion: the gradients of the iterates converge to zero. We notice that the objective function F is lower bounded in most machine learning problems. Even if (U1) and (U2) happen, classical convergence results do guarantee that $\{\nabla F(\theta_t)\} \to 0$ if F is lower bounded. For many practitioners, this guarantee is already good enough.

Convergence theorems.

There are mainly two types of convergence results for gradient descent. Proposition 1.2.1 in [24] applies to the minimization of any differentiable function, but it requires line search that is rarely used in large-scale neural network training, so we ignore it. A result more well-known in machine learning area requires Lipschitz smooth gradient. Proposition 1.2.3 in [24] states that if $\|\nabla F(w) - \nabla F(v)\| \le \beta\|w - v\|$ for all w, v, then for GD with constant stepsize less than $2/\beta$, every limit point is a stationary point; further, if the function value is lower bounded, then the gradient converges to 0 [4]. These theorems require the existence of a global Lipschitz constant β of the gradient. However, for neural network problem (20) a global Lipschitz constant does not exist, thus there is a gap between the theoretical assumptions and the real problems. Is there a simple way to fix this gap?

Unfortunately, for rigorous theoreticians, there seems to be no simple way to fix this gap. The lack of global Lipschitz constants is a general challenge for non-linear optimization, and we refer

[3] In logic, the statement "every element of the set A belongs to the set B" does not imply the set A is non-empty; if the set A is empty, then the statement always holds. For example, "every dragon on the earth is green" is a correct statement, since no dragon exists.

[4] The convergence of gradient is not stated explicitly in Proposition 1.2.3 of [24], but is straightforward to derive based on the proofs.

interested readers to Appendix A for a more in-depth discussion. For practitioners, the following claim may be enough for a conceptual understanding of the convergence theory: if all iterates are bounded, then GD with a proper constant stepsize converges [5]. Bounded Lipschitz constants only help the convergence of the generated sequence, but do not guarantee fast convergence speed. A more severe issue for the Lipschitz constant is that it may be exponentially large or exponentially small even if it is bounded. In the next section, we will focus on a closely related issue of gradient explosion/vanishing.

4 Neural-net Specific Tricks

Without any prior experience, training a neural network to achieve a reasonable accuracy can be rather challenging. Nowadays, after decades of trial and research, people can train a large network relatively easily (at least for some applications such as image classification). In this section, we will describe some main tricks needed for training a neural network.

4.1 Possible Slow Convergence Due to Explosion/Vanishing

The most well-known difficulty of training deep neural-nets is probably gradient explosion/vanishing. A common description of gradient explosion/vanishing is from a signal processing perspective. Gradient descent can be viewed as a feedback correction mechanism: the error at the output layer will be propagated back to the previous layers so that the weights are adjusted to reduce the error. Intuitively, when signal propagates through multiple layers, it may get amplified at each layer and thus explode, or get attenuated at each layer and thus vanish. In both cases, the update of the weights will be problematic.

We illustrate the issue of gradient explosion/vanishing via a simple example of 1-dimensional problem:

$$\min_{w_1,w_2,\ldots,w_L\in\mathbb{R}} F(w) \triangleq 0.5(w_1w_2\ldots w_L-1)^2. \tag{13}$$

The gradient over w_i is

$$\nabla_{w_i}F = w_1\ldots w_{i-1}w_{i+1}\ldots w_L(w_1w_2\ldots w_L-1) = w_1\ldots w_{i-1}w_{i+1}\ldots w_Le, \tag{14}$$

where $e = w_1w_2\ldots, w_L-1$ is the error. If all $w_j=2$, then the gradient has norm $2^{L-1}|e|$ which is exponentially large; if all $w_j=1/2$, then the gradient has norm $0.5^{L-1}e$ which is exponentially small.

Example: $F(w)=(w^7-1)^2$, where $w\in\mathbb{R}$ (similar to the example analyzed in [190]). This is a simpler version of (13). The plot of the function is provided in Figure 2. The region $[-1+c, 1-c]$ is flat, which corresponds to vanishing gradient (here c is a small constant, e.g. 0.2). The regions

[5] This statement is somewhat strange from a theoretical perspective, since we do not know a priori whether the iterates are bounded. However, an assumption of bounded iterates is common in optimization literature.

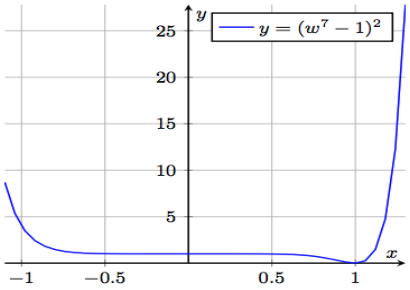


Figure 2: Plot of the function $F(w) = (w^7 - 1)^2$, which illustrates the gradient explosion/vanishing issues. In the region $[-0.8, 0.8]$, the gradients almost vanish; in the region $[1.2, \infty]$ and $[-\infty, -0.8]$, the gradients explode.

$[1 + c, \infty]$ and $[-\infty, -1 - c]$ are steep, which correspond to exploding gradient. Near the global minimum $w = 1$, there is a good basin that if initializing in this region GD can converge fast. If initializing outside this region, say, at $w = -1$, then the algorithm has to traverse the flat region with vanishing gradients which takes a long time. This is the main intuition behind [190] which proves that it takes exponential time (exponential in the number of layers L) for GD with constant stepsize to converge to a global minimum if initializing near $w_i = -1, \forall i$.

Theoretically speaking, why is gradient explosion/vanishing a challenge? This 1-dimensional example shows that gradient vanishing can make GD with constant stepsize converge very slowly. In general, the major drawback of gradient explosion/vanishing is the slow convergence, due to a large condition number and difficulty in picking a proper step-size.

We remark that gradient explosion and vanishing is often considered to be a more severe issue for recurrent neural network (RNN) than for feed-forward neural networks (see, e.g., Goodfellow et al. [77] Sec. 8.2.5), because the same weight matrix is re-used across layers. Another remark is that many works do not mention gradient explosion, but just mention gradient vanishing. This is partially because the non-linear activation function can reduce the signal, and partially because empirical tricks such as "gradient clipping" (simply truncating large values in the gradient) can handle gradient explosion to some extent.

How to resolve the issue of gradient explosion/vanishing? For the 1-dimensional example discussed above, one can choose an initial point inside the basin near the global minimum. Similarly, for a general high-dimensional problem, one solution is to choose an initial point inside a "good basin" that allows the iterates move fast.

In the next subsection, we will discuss initialization strategies in detail.

4.2 Careful Initialization

In the rest of this section, we will discuss three major tricks for training deep neural networks. In this subsection, we discuss the first trick: careful initialization.

As discussed earlier, exploding/vanishing gradient regions indeed exist and occupy a large portion of the whole space, and initializing in these regions will make the algorithm fail. Thus, a

natural idea is to pick the initial point in a nice region to start with.

Naive Initialization Since the "nice region" is unknown, the first thought is to try some simple initial points. One choice is the all-zero initial point, and another choice is a sparse initial point that only a small portion of the weights are non-zero. Yet another choice is to draw the weights from certain random distribution. Trying these initial points would be painful as it is not easy to make them always work: even if an initialization strategy works for the current problem, it might fail for other neural network problems. Thus, a principled initialization method is needed.

LeCun initialization In an early work, [113] proposed to initialize a neural network with sigmoid activation functions as follows:

$$E(W_{ij}^l) = 0, \quad \text{var}(W_{ij}^l) = \frac{1}{d_{l-1}}, \quad l = 1, 2, \ldots, L; i = 1, \ldots, d_{l-1}; j = 1, \ldots, d_l. \tag{15}$$

In other words, the variance of each weight is 1/fan-in, where fan-in is the number of weights fed into the node. Although simple, this is a non-trivial finding. It is not hard to tune the scaling of the random initial point to make it work, but one may find that one scaling factor does not work well for another network. It requires some understanding of neural-nets to realize that adding the dependence on fan-in can lead to a tuning-free initial point. A simple toy experiment can verify the effectiveness of LeCun initialization: compute the ratio $\|\Pi_{l=1}^L W^l x\|/\|x\|$ for $x = (1; 1; \ldots; 1) \in \mathbb{R}^{d\times 1}$ and a random $W \in \mathbb{R}^{d\times d}$ with variance c. When $d > 10L$ and $c = 1/\sqrt{d}$, the ratio is close to 1; when $d > 10L$ and $c = 5/\sqrt{d}$ or $0.2/\sqrt{d}$, the ratio is very large or small.

A theoretical derivation is as follows. Consider a linear neuron with m input $x_1, \ldots, x_m$ and one output $y = \sum_j w_j x_i$. Assume the input x_i has zero mean and variance 1, then y has zero mean and variance $\sqrt{\sum_{j=1}^m w_j^2}$. To make sure the variance of the output is also 1, we only need to pick the weights so that $\text{var}(w_j) = 1/m$ and $E[w_j] = 0$. The above derivation is for linear activations. If the neuron uses the tanh activation $\phi(t) = \tanh(t) = \frac{2}{1+e^{-2t}} - 1$, the gradient $\phi'(t) = \frac{-4e^{-t}}{(1+e^{-t})^2}$ will be around 1 in the "linear regime". Thus $y = \tanh(\sum_j w_j x_j)$ would have variance approximately equal to 1.

Pre-training and Xavier initialization. In late 2000's, the revival of neural networks was attributed to per-training methods that provide good initial point (e.g. [89, 60]). Partially motivated by this trend, Xavier Glorot and Bengio [75] analyzed signal propagation in deep neural networks at initialization, and proposed an initialization method known as Xavier initialization (or Glorot initialization, Glorot normalization):

$$E(W_{ij}^l) = 0, \quad \text{var}(W_{ij}^l) = \frac{2}{d_{l-1} + d_l}, \quad l = 1, 2, \ldots, L; i = 1, \ldots, d_{l-1}; j = 1, \ldots, d_l, \tag{16}$$

or sometimes written as $\text{var}(W_{ij}) = 2/(\text{fan-in} + \text{fan-out})$, where fan-in and fan-out are the input/output dimensions. One example is a Gaussian distribution $W_{ij}^l \sim \mathcal{N}(0, \frac{2}{d_{l-1}+d_l})$, and another example is a uniform distribution $W_{ij}^l \sim \text{Unif}[-\frac{\sqrt{6}}{\sqrt{d_{l-1}+d_l}}, \frac{\sqrt{6}}{\sqrt{d_{l-1}+d_l}}]$.

Xavier initialization can be derived as follows. For feed-forward signal propagation, according to the same argument as LeCun initialization, one could set the variance of the weights to be 1/fan-in.

For the backward signal propagation, according to (10), $e^l = (W^{l+1})^T e^{l+1}$ for a linear network. By a similar argument, one could set the variance of the weights to be 1/fan-out. To handle both feedforward and backward signal propagation, a reasonable heuristic is to set $E(w) = 0, \text{var}(w) = 2/(\text{fan-in} + \text{fan-out})$ for each weight, which is exactly (16).

Kaiming initialization. LeCun and Xavier initialization were designed for sigmoid activation functions which have slope 1 in the "'linear regime" of the activation function. ReLU (rectified linear units) activation [76] became popular after 2010, and He et al. [87] noticed that the derivation of Xavier initialization can be modified to better serve ReLU [6]. The intuition is that for a symmetric random variable ξ, $E[\text{ReLU}(\xi)] = E[\max\{\xi, 0\}] = \frac{1}{2}E[\xi]$, i.e., ReLU cuts half of the signal on average. Therefore, they propose a new initialization method

$$E(W_{ij}^l) = 0, \quad \text{var}(W_{ij}^l) = \frac{2}{d_{\text{in}}} \text{ or } \text{var}(W_{ij}^l) = \frac{2}{d_{\text{out}}}. \tag{17}$$

Note that Kaiming initialization does not try to balance both feedforward and backward signal propagation like Xavier initialization, but just balances one. A recent work [47] discussed this issue, and proposed and analyzed a geometrical averaging initialization $\text{var}(w) = c/\sqrt{(\text{fan-in}) \cdot (\text{fan-out})}$ where c is certain constant.

LSUV. Mishkin and Matas [147] proposed layer-sequential unit-variance (LSUV) initialization that consists of two steps: first, initialize the weights with orthogonal initialization (e.g., see Saxe et al. [184]), then for each mini-batch, normalize the variance of the output of each layer to be 1 by directly scaling the weight matrices. It shows empirical benefits for some problems.

Infinite width networks with general non-linear activations. The derivation of Kaiming initialization cannot be directly extended to general non-linear activations. Even for one dimensional case where $d_i = 1, \forall i$, the output of 2-layer neural network $\hat{y} = \phi(w_2\phi(w_1 x))$ for random weights $w_1, w_2 \in \mathbb{R}$ is a complicated random distribution. To handle this issue, Poole et al. [172] proposed to use mean-field approximation to study infinite-width networks. Roughly speaking, based on the central limit theorem that the sum of a large number of random variables is approximately Gaussian, the pre-activations of each layer are approximately Gaussians, and then they study the evolution of the variance of each layer.

More specifically, for a given input $x \in \mathbb{R}^{d_0}$ and independent random weights $W^1, \ldots, W^L$, the pre-activation at each layer $h^l = (h_1^l, \ldots, h_{d_l}^l)$ are random variables. Notice that $h_i^l = \sum_{j=1}^{d_{l-1}} W_{ij}^l \phi(h_j^{l-1})$ is the weighted sum of d_{l-1} independent zero-mean random variables W_{ij}^l with weights $\phi(h_j^{l-1})$. As the weights $\phi(h_j^{l-1})$ depend on previous layer weights and thus are independent of W_{ij}^l, one can view the weights as "fixed". As the number d_{l-1} goes to infinity, according to central limit theorem, h_i^l will converge to a Gaussian distribution with zero mean and a certain variance, denoted as q^l. For $l \geq 2$, this variance can be computed recursively as

$$q^l = \sigma_w^2 \mathbb{E}_{\xi \sim \mathcal{N}(0,1)}[\phi(\sqrt{q^{l-1}}\xi)^2] = \sigma_w^2 \int \phi(t\sqrt{q^{l-1}})^2 \frac{1}{2\pi} \exp(-\frac{t^2}{2}) dt, l = 2, \ldots, L, \tag{18}$$

[6] Interestingly, ReLU was also popularized by Glorot et al. [76], but they did not apply their own principle to the new neuron ReLU.

where we assume $E(W_{ij}^l) = 0$ and $\text{var}(W_{ij}^l) = \frac{1}{d_{l-1}}\sigma_w^2$. The initial value q^1 is the variance of $h_i^1 = \sum_{j=1}^{d_0} W_{ij}^l x_j$, which can be computed as $q^1 = \sigma_w^2 q^0 = \sigma_w^2 \frac{1}{d_0}\sum_{i=1}^{d_0} x_i^2$, where $q^0 = \frac{1}{d_0}\sum_{i=1}^{d_0} x_i^2$. To check whether q^l computed in such way matches a practical network with finite width, we can compare q^l with the empirical variance $\hat{q}^l = \frac{1}{d_l}\sum_{i=1}^{d_l}(h_i^l)^2$ to see how close they are.

The evolution equation (18) can be used to guide the design of initialization. More specifically, for a certain set of σ_w^2, the equation (18) has a non-zero fixed point q^*, and one can solve the equation numerically and then pick $q^1 = q^*$ which is achieved by scaling the input vector such that its norm $\|x\|^2 = q^* d_0/\sigma_w^2$. Note that for general activation, a case-by-case study of how to pick the initialization variance σ_w^2 and the corresponding input norm $\|x\|$ is needed. See details in Section 2 of [172]. The practical benefit of such a delicate choice of initial variance is not clear.

The above discussion is for the bias-free network and only considers the feedforward signal propagation. A complete analysis includes the variance of the initial bias as an extra degree of freedom, and the backward signal propagation as an extra equation [7]. See more details in [172] and [169].

Finite width networks. The analysis of infinite-width networks can explain the experiments on very wide networks, but narrow networks may exhibit different behavior. Simple experiments show that the output signal strength will be far from the input signal strength if the network is narrow (e.g. when $d = L$ in the toy experiment described earlier).

A rigorous quantitative analysis is given in Hanin and Rolnick [84], which analyzed finite width networks with ReLU activations. The quantity of interest is $M_l = \frac{1}{d_l}\|z^l\|^2, l = 1, \ldots, L$, the normalized post-activation length (this is very similar to $\hat{q}^l$ considered in Poole et al. [172]). The previous analysis of [75] and [87] is concerned about the failure mode that the expected output signal strength $\mathbb{E}[M_L]$ explodes or vanishes. [84] analyzed another failure mode that the empirical variance across layers $\hat{V} \triangleq \frac{1}{L}\sum_{j=1}^{L} M_j^2 - \left(\frac{1}{L}\sum_{j=1}^{L} M_j\right)^2$ explodes or vanishes. They show that with Kaiming initialization (each weight is a zero-mean random variable with variance 2/fan-in), the expectation of the empirical variance $\mathbb{E}[\hat{V}]$ is roughly in the order of $\exp\left(\sum_{k=1}^{L}\frac{1}{d_k}\right)$. If all layers have the same width d, then $\mathbb{E}[\hat{V}]$ is in the order of $\exp(L/d)$. Therefore, for fixed width d, increasing the depth L can make the signal propagation unstable. This might be helpful for explaining why training deep networks is difficult (note that there are other conjectures on the training difficulty of deep networks; e.g. [162]).

Dynamical isometry. Another line of research that aims to understand signal propagation is based on the notion of dynamical isometry [184]. It means that the input-output Jacobian (defined below) has *all* singular values close to 1. Consider a neural-net $f(x) = \phi(W^L\phi(W^{L-1}\ldots\phi(W^1 x)))$, which is slightly different from (1) (with an extra ϕ at the last layer). Its "input-output Jacobian"

[7] We want to remind the readers that Poole et al. [172] derived this extra equation by considering the propagation of the covariance of two inputs (see Section 3 of [172]), while the same equation is presented based on the backward signal propagation in a later paper Pennington et al. [169]. Here we recommend viewing the extra equation as backward signal propagation. Interestingly, the NTK paper [94] reviewed later computes the propagation of the covariance of two inputs as well.

is

$$\frac{\partial z^L}{\partial z^0} = \Pi_{l=1}^{L}(D^l W^l),$$

where D^l is a diagonal matrix with entries being the elements of $\phi'(h_1^l, \ldots, h_{d_l}^l)$. If all singular values of J are close to 1, then according to (10), the back-propagated error $e^l, l = 1, \ldots, L$ will be of similar strength.

Achieving isometry in deep linear networks with equal width (i.e. all d_l's are the same) is very simple: just picking each W^l to be an orthogonal matrix, then their product $W^L W^{L-1} \ldots W^1$ is an orthogonal matrix and thus has all singular values being exactly 1. Saxe et al. [184] showed empirically that for deep linear networks, this orthogonal initialization leads to depth-independent training time, while Gaussian initialization cannot achieve depth-independent training time. This seems to indicate that orthogonal initialization is better than Gaussian initialization, but for non-linear networks this benefit was not observed.

Later, a formal analysis for deep non-linear networks with infinite width was provided in Pennington et al. [169, 170]. They used tools from free probability theory to compute the distribution of all singular values of the input-output Jacobian (more precisely, the limiting distribution as the width goes to infinity). An interesting discovery is that dynamical isometry can be achieved when using sigmoid activation and orthogonal initialization, but cannot be achieved for Gaussian initialization. Note that one needs to carefully pick σ_w^2, σ_b^2 and $\|x\|^2$, and simply using orthogonal initialization is not enough, which partially explains why Saxe et al. [184] did not observe the benefit of orthogonal initialization.

Dynamical isometry for other networks. One obstacle of applying orthogonal initialization to practical networks is convolution operators: it is not clear at all how to compute an "orthogonal" convolution operator. Xiao et al. [226] further studied how to achieve dynamical isometry in deep CNN. They proposed two orthogonal initialization methods for CNN (the simpler version is called DeltaOrthogonal), with which they can train a 10000-layer CNN without other tricks like batch-normalization or skip connections (these tricks are discussed later). This indicates that for training ultra-deep networks, carefully chosen initialization is enough (note that the test accuracy on CIFAR10 is not as good as state-of-the-art perhaps due to the limited representation power around that initial point).

The analysis of dynamical isometry has been applied to other neural networks as well. Li and Nguyen [120] analyzed dynamical isometry for deep autoencoders, and showed that it is possible to train a 200-layer autoencoder without tricks like layer-wise pre-training and batch normalization. Gilboa et al. [74] analyzed dynamical isometry for LSTM and RNNs, and proposed a new initialization scheme that performs much better than traditional initialization schemes in terms of reducing training instabilities.

Computing spectrum. Empirically computing the spectrum (of certain matrices) is very useful for understanding the training process. Dynamical isometry is about the input-output Jacobian, and there are a few other matrices that have been studied. Sagun et al. [180, 181] plotted the distribution of eigenvalues of the Hessian for shallow neural networks. They observed a few

outlier eigenvalues that are a few orders of magnitudes larger than other eigenvalues. Computing the eigenvalues for large networks is very time consuming. To tackle this challenge, Ghorbani et al. [73] used stochastic Lanczos quadrature algorithm to estimate the spectrum density for large-scale problems such as 32-layer ResNet on ImageNet. They confirmed the finding of outlier eigenvalues of [181] for ImageNet. Besides the details of the eigenvalue distributions, these numerical findings indeed verify that the local Lipschitz constant of the gradient (the maximum eigenvalue of the Hessian) is rather small in practical neural network training, partially due to careful initialization.

Along a different line, Brock et al. [31] calculated the top three eigenvalues of the weight matrices (not the Hessian) to track the training process of generative adversarial networks. In a convolutional neural network, the weight is actually a tensor, and Brock et al. [31] reshaped it into a matrix and computes the spectrum of this matrix. Sedghi et al. [188] provided a simple formula to exactly compute the singular values of the linear transformation of each layer, which is defined by a convolution operator.

4.3 Normalization Methods

The second approach is normalization during the algorithm. This can be viewed as an extension of the first approach: instead of merely modifying the initial point, this approach modifies the network for all the following iterates. One representative method is batch normalization (BatchNorm) [92], which is a standard technique nowadays.

Preparation: data normalization. To understand BatchNorm, let us first review a common data preprocessing trick: for linear regression problem $\min_w \sum_{i=1}^n (y_i - w^T x_i)^2$, we often scale each row of the data matrix $[x_1, x_2, \ldots, x_n] \in \mathbb{R}^{d_x \times n}$ so that each row has zero mean and unit norm (one row corresponds to one feature). This operation can be viewed as a pre-conditioning technique that can reduce the condition number of the Hessian matrix, which increases the convergence speed of gradient-based methods.

Motivation of BatchNorm: layerwise normalization. How to extend this idea to deep neural-nets? Intuitively, the convergence speed of each weight matrix W^l is related to the "input matrix" to that layer, which is the matrix of pre-activations $[h^l(1), h^l(2), \ldots, h^l(n)]$, where $h^l(k)$ represents the pre-activation at the l-th layer for the k-th sample (h^l is defined in 8). Thus it is natural to hope that each row of $[h^l(1), h^l(2), \ldots, h^l(n)]$ has zero mean and unit variance. To achieve the extra goal, a naive method is to normalize the pre-activation matrix after updating all weights by a gradient step, but it may ruin the convergence of GD.

Essence of BatchNorm. The solution of [92] is to view this normalization step as a nonlinear transformation "BN" and add BN layers to the original neural network. BN layers play the same role as the activation function ϕ and other layers (such as max pooling layers). This modification can be consistent with BP as long as the chain rule of the gradient can be applied, or equivalently, the gradient of this operation BN can be computed. Note that a typical optimization-style solution would be to add constraints that encode the requirements; in contrast, the solution of BN is to add a non-linear transformation to encode the requirements. This is a typical neural-net style solution.

More details of BatchNorm are given in Appendix B.

Understanding BatchNorm. The original BatchNorm paper claims that BatchNorm reduces the "internal covariate shift". Santurkar et al. [183] argues that internal covariate shift has little do with the success of BatchNorm, and the major benefit of BatchNorm is to reduce the Lipschitz constants (of the objective and the gradients). Bjorck et al. [26] shows that the benefit of BatchNorm is to allow larger learning rate, and discusses the relation with initialization schemes. Arora et al. [11], Cai et al. [35], Kohler et al. [108] analyzed the theoretical benefits of BatchNorm (mainly larger or auto-tuning learning rate) under various settings. Ghorbani et al. [73] numerically found that for networks without BatchNorm, there are large isolated eigenvalues, while for networks with BatchNorm this phenomenon does not occur.

Other normalization methods. One issue of BatchNorm is that the mean and the variance for each mini-batch is computed as an approximation of the mean/variance for all samples, thus if different mini-batches do not have similar statistics then BN does not work very well. Researchers have proposed other normalization methods such as weight normalization [182], layer normalization [13], instance normalization [211], group normalization [225] and spectral normalization [148] and switchable normalization [136].

These methods can be divided into two classes. The first class of methods normalize the intermediate outcome of the neural network (often the pre-activations). For a pre-activation matrix $(h(1), \dots, h(n))$ at a certain layer (we ignore the layer index), BatchNorm chooses to normalize the rows (more precisely, divide each row into many segments and normalize each segment), layer normalization normalizes the columns, and group normalization normalizes a sub-matrix that consists of a few columns and a few rows.

The second class of methods directly normalize the weight matrices. Weight normalization [182] reparameterizes a weight vector w as $g\frac{v}{\|v\|}$, i.e. separates the norm and the direction of the weight matrix, and solve a new problem with g and v being new parameters to learn. Spectral normalization [148] changes the weight matrix W to $\mathrm{SN}(W) = \frac{W}{\sigma_{\max}(W)}$ where $\sigma_{\max}(W)$ is the spectral norm of W, and considers a new neural network $f_\theta(x) = \mathrm{SN}(W^L)\phi(\mathrm{SN}(W^{L-1})\dots\mathrm{SN}(W^2)\phi(\mathrm{SN}(W^1)x)\dots)$. Some of these normalization methods can outperform BatchNorm in a few scenarios, such as RNNs [13], problems where only small mini-batches are available [225] and generative adversarial networks [148].

4.4 Changing Neural Architecture

The third approach is to change the neural architecture. Around 2014, people noticed that from AlexNet [109] to Inception [205], the neural networks get deeper and the performance gets better, thus it is natural to further increase the depth of the network. However, even with smart initialization and BatchNorm, people found training more than 20-30 layers is very difficult. As shown in [88], for a given network architecture VGG, a 56-layer network achieves worse training and test

accuracy than a 20-layer network [8]. Thus, a major challenge at that time was to make training an "ultra-deep" neural network possible.

ResNet. The key trick of ResNet [88] is simple: adding an identity skip-connection for every few layers. More specifically, ResNet changes the network from (2) to

$$z^0 = x; z^l = \phi(\mathcal{F}(W^l, z^{l-1}) + z^{l-1}), \quad l = 1, \dots, L, \tag{19}$$

where $\mathcal{F}$ represents a few layers of the original networks, such as $\mathcal{F}(W_1, W_2, z) = W_1\phi(W_2 z)$. Note that a commonly seen expression of ResNet (especially in theoretical papers) is $z^l = \mathcal{F}(W^l, z^{l-1}) + z^{l-1}$, which does not have the extra $\phi(\cdot)$, but (19) is the form used in practical networks. Note that the expression (19) only holds when the input and output have the same dimension; to change the dimension across layers, one could use extra projection matrices (i.e. change the second term z^{l-1} to $U^l z^{l-1}$) or use other operations (e.g. pooling). In theoretical analysis, the form of (19) is often used.

ResNet has achieved remarkable success: with the simple trick of adding identity skip connection (and also BatchNorm), ResNet with 152 layers greatly improved the best test accuracy at that time for a few computer vision tasks including ImageNet classification (improving top-5 error to a remarkable 3.57%).

Other architectures. Neural architecture design is one of the major threads of current deep learning research. Other popular architecture related to ResNet include high-way networks [201], DenseNet [91] and ResNext [227]. While these architectures are designed by humans, another recent trend is the automatic search of neural architectures (neural architecture search) [254]. There are also intermediate approaches: search one or few hyper-parameters of the neural-architecture such as the width of each layer [237, 207]. Currently, the state-of-the-art architectures (e.g. EfficientNet [207]) for ImageNet classification can achieve much higher top-1 accuracy than ResNet (around 85% v.s. 78 %) with the aid of a few extra tricks.

Analysis of ResNet and initialization. Understanding the theoretical advantage of ResNet or skip connections has attracted much attention. The benefits of skip connections are likely due to multiple factors, including better generalization ability (or feature learning ability), better signal propagation and better optimization landscape. For instance, Orhan and Pitkow [162] suggests that skip connections improve the landscape by breaking symmetry.

Following the theme of this section on signal propagation, we discuss some results on the signal propagation aspects of ResNet. As mentioned earlier, Hanin [83] discussed two failure modes for training; in addition, it proved that for ResNet if failure mode 1 does not happen then failure mode 2 does not happen either. Tarnowski et al. [208] proved that for ResNet, dynamic isometry can be achieved for any activation (including ReLU) and any bi-unitary random initialization (including Gaussian and Orthogonal initialization). In contrast, for the original (non-residual) network, dynamic isometry is achieved only for orthogonal initialization and certain activations (excluding ReLU).

[8] Note that this difficulty is probably not due to gradient explosion/vanishing, and perhaps related to singularities [162].

Besides theoretical analysis, some works further explored the design of new initialization schemes such as [231, 15, 245]. Yang and Schoenholz [231] analyzed randomly initialized ResNet and showed that the optimal initial variance is different from Xavier or He initialization and should depend on the depth. Balduzzi et al. [15] analyzed ResNet with recursion $z^{l+1} = z^l + \beta W^l \cdot \mathrm{ReLU}(z^l)$, where β is a scaling factor. It showed that for β-scaled ResNet with BatchNorm and Kaiming intialization, the correlation of two input vectors scales as $\frac{1}{\beta\sqrt{L}}$, thus it suggests a scaling factor $\beta = 1/\sqrt{L}$. Zhang et al. [245] analyzed the signal propagation of ResNet carefully, and proposed Fixup initialization which leads to good performance on ImageNet, without using BatchNorm. This is probably the first such good result on ImageNet without normalization methods. It modifies Kaiming initialization in the following ways: first, scale all weight layers inside residual branches by $L^{-1/(2m-2)}$, where m is the depth of each residual branch and L is the number of "residual layers" (e.g. for ResNet50, $m = 3$, $L = 16$); second, set the last layer of each residual branch to 0; third, add a scalar multiplier and bias to various layers.

The major modification of Fixup initialization is the scaling factor $L^{-1/(2m-2)}$, and the intuition can be understood by the following simple examples. Consider a linear 1-dimensional ResNet $y = (1 + w_L) \dots (1 + w_2)(1 + w_1)x$ where the scalars $w_i \sim \mathcal{N}(0, c)$. To ensure that $\|y\|/\|x\| \approx O(1)$, we need to pick $c \leq 1/L$. Note that if $c \approx 0$, of course $\|y\|/\|x\| \approx 1$, but then the network has little representation power, thus we want to pick c as large as possible, such as $c = 1/L$. This explains the part of L^{-1} in the scaling factor. Consider another 1-dimensional ResNet $y = (1 + u_m^L \dots u_2^L u_1^L) \dots (1 + u_m^1 \dots u_2^1 u_1^1)x$, where each residual branch has m layers. We want $\mathrm{var}(u_m^i \dots u_2^i u_1^i) = 1/L$, thus it is natural to choose $\mathrm{var}(u_j^i) = L^{-1/m}$, or similarly, multiplying a standard Gaussian variable by $L^{-1/2m}$. This is very close to the scaling factor $L^{-1/(2m-2)}$ used in Fixup initialization.

4.5 Training Ultra-Deep Neural-nets

There are a few approaches that can currently train very deep networks (say, more than 1000 layers) nowadays to reasonable test accuracy for image classification tasks.

- The most well-known approach uses all three tricks discussed above (or variants): proper initialization, proper architecture (e.g. ResNet) and BatchNorm.
- As mentioned earlier, only using a very carefully chosen initial point [226] is enough for training ultra-deep CNNs (though this work does not achieve the best test accuracy).
- Using FixUp initialization and ResNet [9] [245].

Besides the three tricks discussed in this section, there are quite a few design choices that are probably important for achieving good performance of neural networks. These include but not limited to data processing (data augmentation, adversarial training, etc.), optimization methods (optimization algorithms, learning rate schedule, learning rate decay, etc.), regularization (ℓ_2-norm

[9] Note that this paper also uses a certain scalar normalization trick that is much simpler than BatchNorm.

regularization, dropout, etc.), neural architecture (depth, width, connection patterns, filter numbers, etc.) and activation functions (ReLU, leaky ReLU, ELU, tanh, swish, etc.). We have only discussed three major design choices which are relatively well understood in this section. We will discuss a few other choices in the following sections, mainly the optimization methods and the width.

5 General Algorithms for Training Neural Networks

In the previous section, we discussed neural-net specific tricks. These tricks need to be combined with an optimization algorithm such as SGD, and are largely orthogonal to optimization algorithms. In this section, we discuss optimization algorithms used to solve neural network problems, which are often generic and can be applied to other optimization problems as well.

The goals of algorithm design for neural-net optimization are at least two-fold: first, converge faster; second, improve certain metric of interest. The metrics of interest can be very different from the optimization loss, and is often measured on unseen data. A faster method does not necessarily generalize better, and not necessarily improves the metric of interest. Due to this gap, a common algorithm design strategy is: try an optimization idea to improve the convergence speed, but only accept the idea if it passes a certain "performance check". In this section, we discuss optimization algorithms commonly used in deep learning, which are popular due to both optimization reasons and non-optimization reasons. For a more detailed tutorial of standard methods for machine learning (not just deep learning), see Bottou, Curtis and Nocedal [30] and Curtis and Scheinberg [43]

5.1 SGD and learning-rate schedules

We can write (20) as a finite-sum optimization problem:

$$\min_{\theta} F(\theta) \triangleq \frac{1}{B}\sum_{i=1}^{B} F_i(\theta). \tag{20}$$

Each $F_i(\theta)$ represents the sum of training loss for a mini-batch of training samples (e.g. 32, 64 or 512 samples), and B is the total number of mini-batches (smaller than the total number of training samples n). The exact expression of F_i does not matter in this section, as we only need to know how to compute the gradient $\nabla F_i(\theta)$.

Currently, the most popular class of methods are SGD and its variants. Theoretically, SGD works as follows: at the t-th iteration, randomly pick i and update the parameter by

$$\theta_{t+1} = \theta_t - \alpha_t \nabla F_i(\theta_t).$$

In practice, the set of all samples are randomly shuffled at the beginning of each epoch, then split into multiple mini-batches. At each iteration, one mini-batch is loaded into the memory for computation (computing mini-batch gradient and performing weight update).

Reasons for SGD: memory constraint and faster convergence. The reasons of using SGD instead of GD are the memory constraint and the faster convergence. A single GPU or CPU cannot load all samples into its memory for computing the full gradient, thus loading a mini-batch of samples at each iteration is a reasonable choice. Nevertheless, even with the memory constraint, the original GD can be implemented by accumulating all mini-batch gradients without updating the parameters at each iteration. Compared to GD implemented in this way, the advantage of SGD is the faster convergence speed. We defer a more rigorous description to Section 5.2. We emphasize that SGD is not necessarily faster than GD if all samples can be processed in a single machine in a parallel way, but in the memory-constraint system SGD is often much faster than GD.

How strict is the memory constraint in practice? The number of samples in one mini-batch depends on the size of the memory, and also depends on the number of parameters in the model and other algorithmic requirement (e.g. intermediate output at each layer). For instance, a GPU with memory size 11 Gigabytes can only process 512 samples at one time when using AlexNet for ImageNet, and can only process 64 samples at one time when using ResNet50 for ImageNet [10]. Note that the memory constraint only implies that "processing mini-batches separately" is crucial, but does not imply using gradient methods is crucial. The comparison of SGD over other stochastic methods (e.g. stochastic second-order methods) is still under research; see Section 5.6.

Vanilla learning rate schedules. Similar to the case in general nonlinear programming, the choice of step-size (learning rate) is also important in deep learning. In the simplest version of SGD, constant step-size $\alpha_t = \alpha$ works reasonably well: it can achieve a very small training error and relatively small test error for many common datasets. A more popular version of SGD is to divide the step-size by a fixed constant once every few epochs (e.g. divide by 10 every 5-10 epochs) or divide by a constant when stuck. Some researchers refer to SGD with such simple steps-size update rule as "vanilla SGD".

Learning rate warmup. "Warmup" is a commonly used heuristic in deep learning. It means to use a very small learning rate for a number of iterations, and then increases to the "regular" learning rate. It has been used in a few major problems, including ResNet [88], large-batch training for image classification [80], and many popular natural language architectures such as Transformer networks [212] BERT [49]. See Gotmare et al. [79] for an empirical study of warmup.

Cyclical learning rate. A particularly useful variant is SGD with cyclical learning rate ([195, 132]). The basic idea is to let the step-size bounce between a lower threshold and an upper threshold. In one variant called SGDR (Smith [195]), the general principle is to gradually decrease and then gradually increase step-size within one epoch, and one special rule is to use piecewise linear step-size. A later work [196] reported "super convergence behavior" that SGDR converges several times faster than SGD in image classification. In another variant of Ioshchilov et al. [132], within one epoch the step-size gradually decreases to the lower threshold and *suddenly* increases to the upper threshold ("restart"). This "restart" strategy resembles classical optimization tricks in, e.g., Powell [173] and ODonoghue and Candes [160]. Gotmare et al. [79] studied the reasons of the success of cyclical learning rates, but a thorough understanding remains elusive.

[10] Most implementations of ResNet50 only process 32 samples in one GPU.

5.2 Theoretical analysis of SGD

In the previous subsection, we discussed the learning rate schedules used in practice; next, we discuss the theoretical analysis of SGD. The theoretical convergence of SGD has been studied for decades (e.g., [137]). For a detailed description of the convergence analysis of SGD, we refer the readers to Bottou et al. [30]. However, there are at least two issues of the classical analysis. First, the existing analysis assumes Lipschitz continuous gradients similar to the analysis of GD, which cannot be easily justified as discussed in Section 3.2. We put this issue aside, and focus on the second issue that is specific to SGD.

Constant v.s. diminishing learning rate The existing convergence analysis of SGD often requires diminishing step-size , such as $\eta_t = 1/t^\alpha$ for $\alpha \in (1/2, 1]$ [137, 30]. Results for SGD with constant step-size also exist (e.g., [30, Theorem 4.8]), but the gradient does not converge to zero since there is an extra error term dependent on the step-size. This is because SGD with constant stepsize may finally enter a "confusion zone" in which iterates jump around [137]. Early works in deep learning (e.g. LeCun et al. [113]) suggested using diminishing learning rate such as $O(1/t^{0.7})$, but nowadays constant learning rate works quite well in many cases. For practitioners, this unrealistic assumption on the learning rate makes it harder to use the theory to guide the design of the optimization algorithms. For theoreticians, using diminishing step-size may lead to a convergence rate far from practical performance.

New analysis for constant learning rate: realizable case. Recently, an explanation of the constant learning rate has become increasingly popular: if the problem is realizable (the global optimal value is zero), then SGD with constant step-size does converge [186, 213] [11]. In other words, if the network is powerful enough to represent the underlying function, then the stochastic noise causes little harm in the final stages of training, i.e., realizability has an "automatic variance reduction" effect [129]. Note that "zero global minimal value" is a strong assumptions for a general unconstrained optimization problem, but the purpose of using neural networks is exactly to have strong representation power, thus "zero global minimal value" is a reasonable assumption in deep learning. This line of research indicates that neural network optimization has special structure, thus classical optimization theory may not provide the best explanations for neural-nets.

Acceleration over GD. We illustrate why SGD is faster than GD by a simple realizable problem. Consider a least squares problem $\min_{w\in\mathbb{R}^d} \frac{1}{2n}\sum_{i=1}^n (y_i - w^T x_i)^2$, and assume the problem is realizable, i.e., the global minimal value is zero. For simplicity, we assume $n \geq d$, and the data are normalized such that $\|x_i\| = 1, \forall i$. It can be shown (e.g. [213, Theorem 4]) that the convergence rate of SGD with learning rate $\eta = 1$ is $\frac{n}{d}\frac{\lambda_{\max}}{\lambda_{\text{avg}}}$ times better than GD, where $\lambda_{\max}$ is the maximum eigenvalue of the Hessian matrix $\frac{1}{n}XX^T$ and λ_{avg} is the average eigenvalue of the same matrix. Since $1 \leq \frac{\lambda_{\max}}{\lambda_{\text{avg}}} \leq d$, the result implies that SGD is n/d to n times faster than GD. In the extreme case that all samples are almost the same, i.e., $x_i \approx x_1, \forall\, i$, SGD is about n times faster than GD. In the above analysis, we assume each mini-batch consists of a single sample. When there are N

[11] Rigorously speaking, the conditions are stronger than realizability (e.g. weak growth condition in [213]). For certain problems such as least squares, realizablity is enough since it implies the weak growth condition in [213].

mini-batches, SGD is roughly 1 to N times faster than GD. In practice, the acceleration ratio of SGD over GD depends on many factors, and the above analysis can only provide some preliminary insight for understanding the advantage of SGD.

5.3 Momentum and accelerated SGD

Another popular class of methods are SGD with momentum and SGD with Nesterov momentum. SGD with momentum works as follows: at the t-th iteration, randomly pick i and update the momentum term and the parameter by

$$m_t = \beta m_{t-1} + (1-\beta)\nabla F_i(\theta_t); \quad \theta_{t+1} = \theta_t - \alpha_t m_t.$$

We ignore the expression of SGD with Nesterov momentum (see, e.g., [178]).

They are the stochastic versions of the heavy-ball method and accelerated gradient method, but are commonly rebranded as "momentum methods" in deep learning. They are widely used in machine learning area not only because of faster speed than vanilla SGD in practice, but also because of the theoretical advantage for convex or quadratic problems; see Appendix A for more detailed discussions.

Theoretical advantage of SGD with momentum. The classical results on the benefit of momentum only apply to the batch methods (i.e. all samples are used at each iteration). It is interesting to understand whether momentum can improve the speed of the stochastic version of GD in theory. Unfortunately, even for convex problems, achieving such a desired acceleration is not easy according to various negative results (e.g. [51, 50, 106]). For instance, Kidambi et al. [106] showed that there are simple quadratic problem instances that momentum does not improve the convergence speed of SGD. Note that this negative result of [106] only applies to the naive combination of SGD and momentum terms for a general convex problem.

There are two ways to obtain better convergence rate than SGD. First, by exploiting tricks such as variance reduction, more advanced optimization methods (e.g. [127, 2]) can achieve an improved convergence rate that combines the theoretical improvement of both momentum and SGD. However, these methods are somewhat complicated, and are not that popular in practice. Defazio and Bottou [46] analyzed the reasons why variance reduction is not very successful in deep learning. Second, by considering more structure of the problem, simpler variants of SGD can achieve acceleration. Jain et al. [95] incorporated statistical assumption of the data to show that a certain variant is faster than SGD. Liu and Belkin [128] considered realizable quadratic problems, and proposed a modified version of SGD with Nesterov's momentum which is faster than SGD.

Accelerated SGD for non-convex problems. The above works only apply to convex problems and are thus not directly applicable to neural network problems which are *non-convex*. Designing accelerated algorithms for general non-convex problems is quite hard: even for the batch version, accelerated gradient methods cannot achieve better convergence rate than GD when solving non-convex problems. There have been many recent works that design new methods with faster convergence rate than SGD on general non-convex problems (e.g. [37, 36, 229, 61, 3] and references

therein). These methods are mainly theoretical and not yet used by practitioners in deep learning area. One possible reason is that they are designed for worst-case non-convex problems, and do not capture the structure of neural network optimization.

5.4 Adaptive gradient methods: AdaGrad, RMSProp, Adam and more

The third class of popular methods are adaptive gradient methods, such as AdaGrad [59], RMSProp [210] and Adam [107]. We will present these methods and discuss their empirical performance and the theoretical results.

Descriptions of adaptive gradient methods. AdaGrad works as follows: at the t-th iteration, randomly pick i, and update the parameter as (let $\circ$ denote entry-wise product)

$$\theta_{t+1} = \theta_t - \alpha_t v_t^{-1/2} \circ g_t, \quad t = 0, 1, 2, \ldots, \tag{21}$$

where $g_t = \nabla F_i(\theta_t)$ and $v_t = \sum_{j=1}^t g_j \circ g_j$. In other words, the step-size for the k-th coordinate is adjusted from α_t in standard SGD to $\alpha_t / \sqrt{\sum_{j=0}^t g_{j,k}^2}$ where $g_{j,k}$ denotes the k-th entry of g_j. AdaGrad can be also written in the form of a stochastic diagonally scaled GD as

$$\theta_{t+1} = \theta_t - \alpha_t D_t^{-1} g_t,$$

where $D_t = \operatorname{diag}\left(\sum_{j=1}^t g_j g_j^T\right)$ is diagonal part of the matrix formed by the average of the outer product of all past gradients. This can be viewed as a stochastic version of the general gradient method in [24, Section 1.2.1], with a special choice of the diagonal scaling matrix D_t. AdaGrad is shown to exhibit a convergence rate similar to SGD for convex problems [59] and non-convex problems (see, e.g., [39]): when the stepsize is chosen to be the standard diminishing stepsize (e.g. $1/\sqrt{t}$) the iteration complexity is $O(\log T/\sqrt{T})$ (i.e. after T iterations, the error is of the order $1/\sqrt{T}$).

One drawback of AdaGrad is that it treats all past gradients equally, and it is thus natural to use exponentially decaying weights for the past gradients. This new definition of v_t leads to another algorithm RMSProp [210] (and a more complicated algorithm AdaDelta [242]; for simplicity, we only discuss RMSProp). More specifically, at the t-th iteration of RMSProp, we randomly pick i and compute $g_t = \nabla F_i(\theta_t)$, and then update the second order momentum v_t and parameter θ_t as

$$\begin{aligned} v_t &= \beta v_{t-1} + (1-\beta) g_t \circ g_t, \\ \theta_{t+1} &= \theta_t - \alpha_t v_t^{-1/2} \circ g_t. \end{aligned} \tag{22}$$

Adam [107] is the combination of RMSProp and the momentum method (i.e. heavy ball method). At the t-th iteration of RMSProp, we randomly pick i and compute $g_t = \nabla F_i(\theta_t)$, and then update the first order momentum m_t, the second order momentum v_t and parameter θ_t as

$$\begin{aligned} m_t &= \beta_1 v_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t \circ g_t, \\ \theta_{t+1} &= \theta_t - \alpha_t v_t^{-1/2} \circ m_t. \end{aligned} \tag{23}$$

There are a few other related methods in the area, e.g. AdaDelta [242], Nadam [54], and interested readers can refer to [178] for more details.

Empirical use of adaptive gradient methods. AdaGrad was designed to deal with sparse and highly unbalanced data. Imagine we form a data matrix with the data samples being the columns, then in many machine learning applications, most rows are sparse (infrequent features) and some rows are dense (frequent features). If we use the same learning rate for all coordinates, then the infrequent coordinates will be updated too slowly compared to frequent coordinates. This is the motivation to use different learning rates for different coordinates. AdaGrad was later used in many machine learning tasks with sparse data such as language models where the words have a wide range of frequencies [146, 168].

Adam is one of the most popular methods for neural network training nowadays [12]. After Adam was proposed, the common conception was that Adam converges faster than vanilla SGD and SGD with momentum, but generalizes worse. Later, researchers found that (e.g., [222]) well-tuned SGD and SGD with momentum outperform Adam in both training error and test error. Thus the advantages of Adam, compared to SGD, are considered to be the relative insensitivity to hyperparameters and rapid initial progress in training (see, e.g. [104]). Sivaprasad et al. [194] proposed a metric of "tunability" and verified that Adam is the most tunable for most problems they tested.

The claim of the "marginal value" of adaptive gradient methods [222] in year 2017 did not stop the booming of Adam in the next two years. Less tuning is one reason, but we suspect that another reason is that the simulations done in [222] are limited to image classification, and do not reflect the real application domains of Adam such as GANs and reinforcement learning.[13] For these tasks, the generalization ability of Adam might be a less critical issue.

Theoretical results on adaptive gradient methods. Do these adaptive gradient methods converge? Although Adam is known to be convergent in practice and the original Adam paper [107] claimed a convergence proof, it was recently found in Reddi et al. [176] that RMSProp and Adam can be divergent (and thus there is some error in the proof of [107]) even for solving convex problems. To understand the reason of divergence, recall that SGD with constant stepsize α_t may not converge [137], but SGD with diminshing step-size (satisfying a few requirements) converges. In AdaGrad, the "effective" stepsize $\alpha_t/\sqrt{v_t}$ is diminishing and AdaGrad converges, but in Adam and RMSProp the effective stepsize $\alpha_t/\sqrt{v_t}$ is not necessarily diminishing (even if the step-size α_t is decreasing), thus causing divergence. To fix the divergence issue, [176] proposed AMSGrad, which

[12] The paper that proposed Adam [107] achieved phenomenal success at least in terms of popularity. It was posted in arxiv on December 2014; by Aug 2019, the number of citations in Google scholar is 26000; by Dec 2019, the number is 33000. Of course the contribution to optimization area cannot just be judged by the number of citations, but the attention Adam received is still quite remarkable.

[13] For the 8 most cited papers in Google Scholar among those citing the original Adam paper [107], and found that four papers are on GANs (generative adversarial networks) [174, 93, 115, 8], two on deep reinforcement learning [149, 126] and two on language-related tasks [228, 212]. This finding is consistent with the claim in [222] that "adaptive gradient methods are particularly popular for training GANs and Q-learning ...".

changes the update of v_t in Adam to the following:

$$\bar{v}_t = \beta_2 \bar{v}_{t-1} + (1-\beta_2) g_t^2, \quad v_t = \max\{v_{t-1}, \bar{v}_t\}.$$

They also prove the convergence of AMSGrad for convex problems (for diminishing β_1). Empirically, AMSGrad is reported to have somewhat similar (or slightly worse) performance to Adam.

The convergence analysis and iteration complexity analysis of adaptive gradient methods are established for non-convex optimization problems in a few subsequent works [39, 250, 256, 45, 257, 219]. For example, [39] considers a general Adam-type methods where v_t can be any function of past gradients $g_1, \ldots, g_t$ and establishes a few verifiable conditions that guarantee the convergence for non-convex problems (with Lipschitz gradient). We refer interested readers to Barakat and Bianchi [16] which provided a table summarizing the assumptions and conclusions for adaptive gradient methods. Despite the extensive research, there are still many mysteries about adaptive gradient methods. For instance, why it works so well in practice is still largely unknown.

5.5 Large-scale distributed computation

An important topic in neural network optimization is how to accelerate training by using multiple machines. This topic is closely related to distributed and parallel computation (e.g. [25]).

Basic analysis of scaling efficiency. Intuitively, having K machines can speed up training by up to K times. In practice, the acceleration ratio depends on at least three factors: communication time, synchronization time and convergence speed. Ignoring the communication time and synchronization time, the acceleration ratio of K can be achieved in an extreme case that data on different machines do not share common features. In another extreme case where data on different machines are the same, the acceleration ratio is at most 1. In practice, the acceleration ratio often lies in the region $[1, K]$. Deep learning researchers often use "scaling efficiency" to denote the ratio between the acceleration ratio and the number of machines. For instance, if K machines are used and the multi-machine training is $K/2$ times faster than single-machine training, then the scaling efficiency is 0.5. The goal is to achieve a scaling efficiency as close to 1 as possible without sacrificing the test accuracy.

Training ImageNet in 1 hour. Goyal et al. [80] successfully trained ResNet50 (50-layer ResNet) for the ImageNet dataset in 1 hour using 256 GPUs; in contrast, the original implementation in He et al. [88] takes 29 hours using 8 GPUs. The scaling efficiency is $29/32 \approx 0.906$, which is remarkable. Goyal et al. [80] used 8192 samples in one mini-batch, while He et al. [88] only used 256 samples in one mini-batch. Bad generalization was considered to be a major issue for large mini-batches, but [80] argued that optimization difficulty is the major issue. They used two major optimization tricks: first, they scale the learning rate with the size of the mini-batches; second, they use "gradual warmup" strategy that increases the learning rate from η/K gradually to η in the first 5 epochs, where K is the number of machines.

Training ImageNet in minutes. Following Goyal et al. [80], a number of works [197, 1, 99, 145, 234, 230] have further reduced the total training time by using more machines. For example,

You et al. [235] applied layer-wise adaptive rate scheduling (LARS) to train ImageNet with mini-batch size 32,000 in 14 minutes. Yamazaki et al. [230] used warmup and LARS, tried many learning rate decay rules and used label smoothing to train ImageNet in 1.2 minutes by 2048 V100 GPUs, with mini-batch size 81920. Note that all these works train ResNet50 on ImageNet to get validation accuracy between 75% to 77%. Multi-machine computation has also been studied on other tasks. For instance, Goyal et al. [80] also tested Mask R-CNN for object detection, and You et al. [236] studied BERT for language pre-training.

5.6 Other Algorithms

Other learning rate schedules. We have discussed cyclical learning rate and adaptive learning rate. Adaptive stepsize or tuning-free step-size has been extensively studied in non-linear optimization area (see, e.g. Yuan [239] for an overview). One of the representative methods is Barzilai-Borwein (BB) method proposed in year 1988 [19]. Interestingly, in machine learning area, an algorithm similar to BB method was proposed in the same year 1988 in Becker et al. [21] (and further developed in Bordes et al. [28]). This is not just a coincidence: it reflects the fact that the problems neural-net researchers have been thinking are very similar to those of non-linear optimizers. LeCun et al. [114] provided a good overview of the tricks for training SGD, especially step-size tuning based on the Hessian information. Other recent works on tuning-free SGD include Schaul [185], Tan et al. [206] and Orabona [161].

Second order methods. Second-order methods have also been extensively studied in the neural network area. Along the line of classical second-order methods, Martens [140] presented Hessian-free optimization algorithms, which are a class of quasi-Newton methods without explicit computation of an approximation of the Hessian matrix (thus called "Hessian free"). One of the key tricks, based on [167, 187], is how to compute Hessian-vector products efficiently by backpropagation, without computing the full Hessian. Berahas [23] proposed a stochastic quasi-Newton method for solving neural network problems. Another tye of second order method is the natural gradient method [6, 141], which scales the gradient by the empirical Fisher information matrix (based on theory of information geometry [5]). We refer the readers to [141] for a nice interpretation of natural gradient method and the survey [30] for a detailed introduction. A more efficient version K-FAC, based on block-diagonal approximation and Kronecker factorization, is proposed in Martens and Grosse [142].

Competition between second order methods and first-order methods. Adaptive gradient methods actually use second-order information implicitly, and may be characterized as second-order method as well (e.g. in Bottou et al. [30]). Here we still view adaptive gradient methods as first order methods since they only use a diagonal approximation of the Hessian matrix; in contrast, second order methods use a matrix approximation of the Hessian in a certain way. Note that there can be a continuous transition between first and second order methods, dependent on how much second-order information is used.

During the early times when neural-nets did not achieve good performance, some researchers

thought that it is due to the limitation of first-order methods and it may be crucial to develop fast second-order methods. In the recent decade, the trend has reversed and first-order methods have been dominant. Bottou and Bousquet [29] provides some theoretical justification why first-order methods are enough for large-scale machine learning problems. Nowadays some researchers thought second order methods cannot compete with first-order methods since they may overfit, and SGD has some implicit regularization effect for achieving good test performance. Nevertheless, very recently, second order methods showed some promise: Osawa et al. [163] has achieved good test performance on ImageNet using K-FAC (only takes 35 epochs to achieve 75% top-1 accuracy on ImageNet). It is interesting to see whether second order methods can revive in the future.

6 Global Optimization of Neural Networks (GON)

One of the major challenges for neural network optimization is non-convexity. A general non-convex optimization problem can be very difficult to solve due to sub-optimal local minima. The recent success of neural networks suggest that neural-net optimization is far from a worst-case non-convex problem, and finding a global minimum is not a surprise in deep learning noways. There is a growing list of literature devoted to understanding this problem. For simplicity of presentation, we call this subarea "global optimization of neural networks" (GON) [14]. We remark that research in GON was partially reviewed in Vidal et al. [217], but most of the works we reviewed here appear after [217].

The previous two sections mainly focus on "local issues" of training. Section 4 discussed gradient explosion/vanishing, and resolving this issue can ensure the algorithm can move locally. Section 4 discussed the convergence speed, but the limitation is that the results only show convergence to local minima (or stationary points). In this section, we adopt a global view of the optimization landscape. Typical questions include but are not limited to: When can an algorithm converge to global minima? Are there sub-optimal local minima? How to pick an initial point that ensures convergence to global minima? What properties do the optimization landscape have?

6.1 Related areas

Before discussing neural networks, we discuss a few related subareas.

Tractable problems. Understanding the boundary between "tractable" and "intractable" problems has been one of the major themes of optimization area. The most well-known boundary is probably between convex and non-convex problems. However, this boundary is vague since it is also known that many non-convex optimization problems can be reformulated as a convex problem

[14] It is not clear how we should call this subarea. Many researchers use "(provable) non-convex optimization" to distinguish these research from convex optimization. However, this name may be confused with the studies of non-convex optimization that focus on the convergence to stationary points. The name "global optimization" might be confused with research on heuristic methods, while GON is mainly theoretical. Anyhow, let's call it global optimization of neural-nets in this article.

(e.g. semi-definite programming and geometric programming). We guess that some neural-net problems are in the class of "tractable" problems, though the meaning of tractability is not clear. Studying neural networks, in this sense, is not much different in essence from the previous studies of semi-definite programming (SDP), except that a theoretical framework as complete as SDP has not been developed yet.

Global optimization. Another related area is "global optimization", a subarea of optimization which aims to design and analyze algorithms that find globally optimal solutions. The topics include global search algorithms for general non-convex problems (e.g. simulated annealing and evolutionary methods), algorithms designed for specific non-convex problems (possibly discrete) (e.g. [133]), as well as analysis of the structure of specific non-convex problems (e.g. [63]).

Non-convex matrix/tensor factorization. The most related subarea to GON is "non-convex optimization for matrix/tensor factorization" (see, e.g., Chi et al. [40] for a survey), which emerged after 2010 in machine learning and signal processing areas [15]. This subarea tries to understand why many non-convex matrix/tensor problems can be solved to global minima easily. Most of these problems can be viewed as the extensions of matrix factorization problem

$$\min_{X,Y\in\mathbb{R}^{n\times r}} \|M - XY^T\|_F^2, \tag{24}$$

including low-rank matrix completion, phase retrieval, matrix sensing, dictionary learning and tensor decomposition. The matrix factorization problem (24) is closely related to the eigenvalue problem. Classical linear algebra textbooks explain the tractability of the (original) eigenvalue problem by proving directly the convergence of power method, but it cannot easily explain what happens if a different algorithm is used. In contrast, an optimization explanation is that the eigenvalue problem can be solved to global optima because every local-min is a global-min. One central theme of this subarea is to study whether a nice geometrical property still holds for a generalization of (24). This is similar to GON area, which essentially tries to understand the structure of deep non-linear neural-nets that also can be viewed as generalization of (24).

6.2 Empirical exploration of landscape

We first discuss some interesting empirical studies on the optimization landscape of neural networks. Some of the empirical studies like lottery ticket hypothesis have sparked a lot of interests from practitioners as they see potential practical use of landscape studies. Theoretical results will be reviewed mainly in later subsections.

One of the early papers that caught much attention is Dauphin et al. [44], which showed that empirically bad local minima are not found and a bigger challenge is plateaus. Goodfellow et al. [78] plotted the function values along the line segment between the initial point and the converged point, and found that this 1-dimensional plot is similar to a 1-dimensional convex plot which has no bumps. These early experiments indicated that the landscape of a neural-net problem is much nicer than one thought.

[15] Again, it is not clear how to call this subarea. "Non-convex optimization" might be a bit confusing to optimizers.

A few later works provided various ways to explore the landscape. Poggio and Liao [171] gave experiments on the visualization of the evolution of SGD. Li et al. [119] provided visualization of the landscape under different network architecture. Baity-Jesi et al. [14] compared the learning dynamics of neural-nets with glassy systems in statistical physics. Franz et al. [66] and Geiger et al. [72] studied the analogy between the landscape of neural networks and the jamming transition in physics.

6.2.1 Mode connectivity

An exact characterization of a high-dimensional surface is almost impossible, thus goemeters strive to identify simple yet non-trivial properties (e.g. Gauss's curvature). One such property called "mode connectivity" has been found for deep neural networks. In particular, Draxler et al. [55] and Garipov et al. [69] independently found that two global minima can be connected by an (almost) equal-value path. This is an empirical claim, and in practice the two "global minima" refer to two low-error solutions found by training from two random initial points.

A more general optimization property is "connectivity of sub-level sets". If the sub-level set $\{\theta : F(\theta) \leq c\}$ is connected for c being the global minimal value, then any two global minima can be connected via an equal-value path. The connectivity of the sub-level sets was first proved by [67] for 1-hidden layer linear networks, and [67] also empirically verified the connectivity for MINST dataset. The contributions of Draxler et al. [55] and Garipov et al. [69] are that they used stronger path-finding algorithms to validate the connectivity of global minima for CIFAR10 and CIFAR100 datasets. The connectivity for deep neural networks was theoretically justified in Nguyen [156], Kuditipudi et al. [110].

6.2.2 Model compression and lottery ticket hypothesis

Another line of research closely related to the landscape is training smaller neural networks (or called "efficient deep learning"). This line of research has a close relation with GON, and this relation has been largely ignored by both theoreticians and practitioners.

The current neural network models often contain a huge number of parameters (millions or even hundreds of millions). Models for solving ImageNet classification are already large, and recent models for other tasks are even bigger (e.g. BERT [49] and bigGAN [31]). While understanding the benefit of over-parameterzation has been a hot topic (reviewed later), for practitioners it is more pressing to design new methods to train smaller models. Smaller models can be used for resource-constrained hardware (e.g. mobile devices, internet-of-things devices), and also accessible to more researchers. However, typically a much smaller models will lead to significantly worse performance.

Network pruning [82] showed that many large networks can be pruned to obtain a much smaller network while the test accuracy is only dropped little. Nevertheless, in network pruning, the small network often has to inherit the weights from the solution found by training the large network to

achieve good performance, and training a small network from the scratch often leads to significantly worse performance [16].

Frankle and Carbin [64] made an interesting finding that in some cases a good initial point is relatively easy to find. More specifically, for some datasets (e.g. CIFAR10), [64] empirically shows that a large network contains a small subnetwork and a certain "half-random" initial point such that the following holds: training the small network from this initial point can achieve performance similar to the large network. This "semi-random" initial point is found by the following procedure: first, record the random initial point θ^0 for the large network, and train the large network to converge to get θ^*; second, define a mask $\Omega \in \{0,1\}^{|\theta|}$ as $\Omega(k) = 1$ if $|\theta_k^*| > \delta$ and $\Omega(k) = 0$ if $|\theta_k^*| > \delta$, where δ is a certain threshold and θ_k^* denotes the k-th element of θ^*; third, define the new initial point as $\tilde{\theta}^0 = \Omega \circ \theta^0$, and the new small network by discarding those weights with zero values in Ω. In short, the new initial point inherits "random" weights from the original random initial point, but it only keeps a subset of the weights and thus the remaining weights are not independent anymore. The trainable subnetwork (the architecture and the associated initial point together) is called a "winning ticket", since it has won an "initialization lottery". Lottery ticket hypothesis (LTH) states that such a winning ticket always exists. Later work [65] shows that for larger datasets such as ImageNet, the procedure in [64] needs to be modified to find a good initial point. Zhou et al. [251] further studies the factors that lead to the success of the lottery tickets (e.g. they find the signs of the weights are very important). For more discussions on LTH, see Section 3.1 of [151].

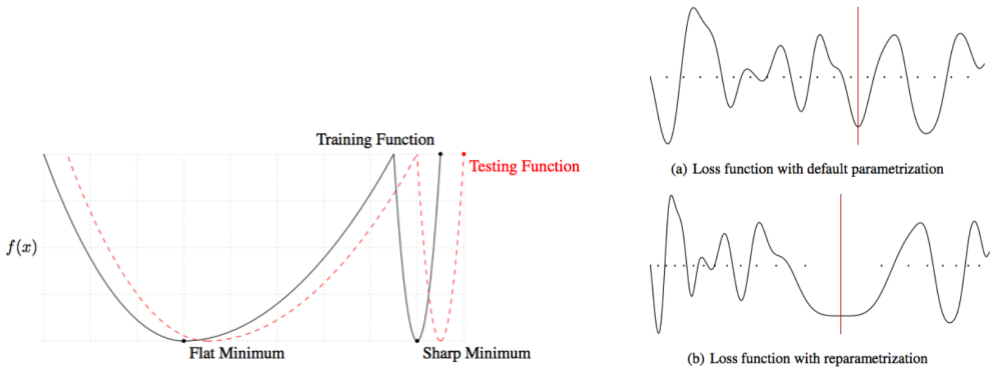
The works on network pruning and LTH are mostly empirical, and a clean message is yet to be stated due to the complication of experiments. It is an interesting challenge to formally state and theoretically analyze the properties related to model compression and LTH.

6.2.3 Generalization and landscape

Landscape has long been considered to be related to the generalization error. A common conjecture is that flat and wide minima generalize better than sharp minima, with numerical evidence in, e.g., Hochreiter and Schmidhuber [90] and Keskar et al. [105]. The intuition is illustrated in Figure 3(a): the test loss function and the training loss function have a small difference, and that difference has a small effect on wide minima and thus they generalize well; in constrast, this small difference has a large effect on sharp minima and thus they do not generalize well. Dinh et al. [53] argues that sharp minima can also generalize since they can become wide minima after re-parameterization; see Figure 3(b). How to define "wide" and "sharp" in a rigorous way is still challenging. Neyshabur et al. [153], Yi et al. [233] defined new metrics for the "flatness" and showed the connection between generalization error and the new notions of "flatness". He et al. [86] found that besides wide and shallow local minima, there are asymmetric minima that the function value changes rapidly along some direction and slowly along some other directions, and algorithms biased towards the wide side

[16] There are some recent pruned networks that can be trained from random initial point [130, 116], but the sparsity level is not very high; see [65, Appendix A] for discussions.

generalize better.



(a) Wide minima generalize better [105]

(b) Sharp minima may become wide after re-parameterization [53]

Figure 3: Illustration on wide minima and sharp minima.

Although the intuition "wide minima generalize better" is debatable, researchers still borrow this intuition to design or discuss optimization algorithms. Chaudhari et al. [38] designed entropy-SGD that explicitly search for wider minima. Smith and Topin [196] also argued that the benefit of cyclical learning rate is that it can escape shallow local minima

6.3 Optimization Theory for Deep Neural Networks

We discuss two recent threads in optimization theory for *deep* neural networks: landscape analysis and gradient dynamics analysis. The first thread discusses the global landscape properties of the loss surface, and the second thread studies gradient dynamics of ultra-wide networks.

6.3.1 Global landscape analysis of deep networks

Global landscape analysis is the closest in spirit to the empirical explorations in Section 6.3: understanding some geometrical properties the landscape. There are three types of deep neural networks with positive results so far: linear networks, over-parameterized networks and modified networks. We will also discuss some negative results.

Deep linear networks. Linear networks have little representation power and are not very interesting from a learning perspective, but it is a valid problem from optimization perspective. The landscape of deep linear networks are relatively well understood. Choromanska et al. [42] uses spin glass theory to analyze deep linear neural-nets (started from ReLU network, but actually analyzed linear network), and proved that local minima have highest chance to be close to global minima among all stationary points (the precise statement is very technical). Kawaguchi [102] proves that for a deep fully-connected linear network with quadratic loss, under mild conditions (certain data

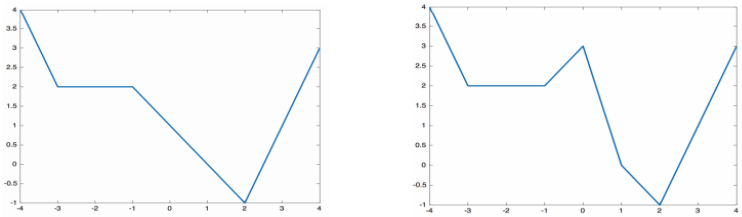


Figure 4: Left figure: the flat region is not a set-wise strict local-min, and this region can be escaped by a (non-strictly) decreasing algorithm. Right figure: there is a basin that is a set-wise strict local-min.

matrices are full-rank and output dimension d_y is no more than input dimension d_x), every local-min is a global-min. Lu and Kawaguchi [134] provides a much simpler proof for this result under stronger conditions. Laurent and James [111] extends this result to arbitrary loss functions, and Zhang [247] gives a further simplified proof. Hardt and Ma [85] analyzes the number of stationary points in a small region around global minima for linear ResNet. Nouiehed and Razaviyayn [158] provided a general sufficient condition for the local-min of a neural-net to be global-min, and apply this condition to deep linear networks (also give a weaker result for non-linear pyramid networks). Besides characterizing local minima, stronger claims on the stationary points can be proved for linear networks. Yun et al. [240] and Zou et al. [253] present necessary and sufficient conditions for a stationary point to be a global minimum.

Deep over-parameterized networks. Over-parameterized networks are the simplest non-linear networks that currently can be analyzed, but already somewhat subtle. It is widely believed that "more parameters than necessary" can smooth the landscape [131, 154, 244], but these works do not provide a rigorous result. To obtain rigorous results, one common assumption for deep networks is that the last layer has more neurons than the number of samples. Under this assumption on the width of the last layer, Nguyen et al. [157] and Li et al. [118] prove that a fully connected network has no "spurious valley" or "set-wise strict local minima", under mild assumptions on the data. The difference is that Nguyen et al. [157] requires the activation functions satisfy some conditions (e.g. strictly increasing) and can extend the result to other connection patterns (including CNN), and Li et al. [118] only requires the activation functions to be continuous (thus including ReLU and swish). Intuitively, "set-wise strict local minima" and "spurious valley" are the "bad basin" illustrated in the right figure of Figure 4 (see [157] or [118] for formal definitions).

The above works are the extensions of a classical work [238] on 1-hidden-layer over-parameterized networks (with sigmoid activations), which claimed to have proved that every local-min is a global-min. It was later found in [118] that the proof is not rigorous. Ding et al. [52] further constructs sub-optimal local-min for arbitrarily wide neural networks for a large class of activations including sigmoid activations, thus under the settings of [238][157] [118] sub-optimal local minima can exist. This implies that overparameterization cannot eliminate bad local minima, but only bad basins (or spurious valleys).

Finally, it seems that over-parameterized networks are prone to over-fitting, but many practical networks are indeed over-parameterized and understanding why over-fitting does not happen is an

interesting line of research [154, 18, 220, 224, 22, 143]. In this article, we mainly discuss the research on the optimization side.

Modified problems. The results discussed so far mainly study the original neural network problem (20), and the landscape is different if the problem is slightly changed. Liang et al. [124] provides two modifications, each of which can ensure no bad local-min exists, for binary classification. Kawaguchi et al. [103] extends the result of [124] to multi-class classification problems. In addition, [103] provides toy examples to illustrate the limitation of only considering local minima: GD may diverge for the modified problem. It is a possible weakness of any result on "no bad local-min" including the classical works on deep linear networks. In fact, as discussed in Section 3.2, the possibility of divergence (U3) is one of the three undesirable situations that classical results on GD does not exclude, and eliminating bad local-min only excludes (U1).

Negative results. Most of the works in GON area after 2012 are positive results. However, while neural-nets can be trained in some cases with careful choices of architecture, initial points and parameters, there are still many cases that neural-nets cannot be successfully trained. Shalev et al. [189] explained a few possible reasons of failure of GD for training neural networks. There are a number of recent works focusing the existence of bad local minima.

These negative results differ by their assumptions on activation functions, data distribution and network structure. As for the activation functions, many works showed that ReLU networks have bad local minima (e.g., Swirszcz et al.[204] Zhou et al. [252], Safran et al.[179], Venturi et al.[216], Liang et al.[125]), and a few works Liang et al. [125], Yun et al.[241] and Ding et al. [52] construct examples for smooth activations. As for the loss function, Safran and Shamir [179] and Venturi et al. [216] analyze the population risk (expected loss) and other works analyze the empirical risk (finite sum loss). As for the data distribution, most works consider data points that lie in a zero-measure space or satisfy special requirements like linear separability (Liang et al. [125]) or Gaussian (Safran et al.[179]), and few consider generic input data (e.g. Ding et al. [52]). We refer the readers to Ding et al. [52] which compared various counter-examples in a table.

6.3.2 Algorithmic analysis of deep networks

A good landscape may make an algorithm easier to find global minima, but does not fully explain the behavior of specific algorithms. To understand specific algorithms, convergence analysis is more desirable. However, for a general neural-net the convergence analysis is extremely difficult, thus some assumptions have to be made. The current local (algorithmic) analysis of deep neural-nets is mainly performed for two types: linear networks [184, 17, 9, 98] and ultra-wide networks.

Linear networks. As discussed earlier, gradient explosion/vanishing can cause great difficulty of training neural-nets, and even for the scalar problem $\min_{w_1,\dots,w_L}(1-w_1\dots w_L)^2$, it takes GD exponential time to converge [190]. Perhaps a bit surprisingly, for deep linear networks in higher dimension, polynomial time convergence can still be established. Arora et al. [9] considered the problem $\min_{W_1,\dots,W_L}\|W_1W_2\dots W_L-\Phi\|_F^2$, and prove that if the initial weights are "balanced" and the initial product $W_1\dots W_L$ is close to Φ, GD with a small stepsize converges to global minima in

polynomial time. Ji and Telgarsky [98] assume linearly separable data and prove that if the initial objective value is less than a certain threshold, then GD with small adaptive stepsize converges asymptotically to global minima. Moreover, they proved that the normalized weight matrices converge to rank-1 matrices, which matches the empirical observation that the converged weight matrices are approximately low rank in AlexNet. The strong assumptions of these works on initialization, small stepsize and/or data are still far from satisfactory, but at least some of these assumptions are necessary in the worst-case (as discussed in [9]). Shin [191] analyzed layerwise-training for deep linear networks, and showed that under some conditions, gradient descent convergences faster for deeper networks.

Neural Tangent Kernal (NTK). Convergence analysis for deep non-linear networks is much harder than linear networks, even under the extra assumption of over-parametrization. Some progress has been made recently. We first discuss the result of Jacot et al. [94] on NTK.

This NTK result is an extension of a property of linear regression. A typical explanation why GD converges to global minima of linear regression is that the objective function is convex, then for neural networks one would extend convexity to other geometrical properties (basically the idea behind landscape analysis). There is another explanation from the perspective of gradient flow. Consider the linear regression problem $\min_{w\in\mathbb{R}^d} F(w) \triangleq \frac{1}{2}\sum_{i=1}^n (w^T x_i - y_i)^2$. The gradient flow is $\frac{dw(t)}{dt} = -XX^T w(t) + Xy$, where $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$. Define $r_i = w^T x_i - y_i, i = 1, \ldots, n$, then the dynamics of the residual $r = (r_1; \ldots; r_n) \in \mathbb{R}^{n\times 1}$ is

$$\frac{dr(t)}{dt} = -X^T X r(t). \tag{25}$$

This is called *kernel gradient descent* with respect to the kernel $K = X^T X \succeq 0$.

Consider the neural-network problem with quadratic loss $\min_\theta \sum_{i=1}^n \frac{1}{2}(f_\theta(x_i) - y_i)^2$, where $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$ (it can be generalized to multi-dimensional output and non-quadratic loss). The gradient descent dynamics is

$$\frac{d\theta}{dt} = -\sum_i \frac{\partial f_\theta(x_i)}{\partial \theta}(f_\theta(x_i) - y_i). \tag{26}$$

Define $G = (\frac{\partial f_\theta(x_1)}{\partial \theta}, \ldots, \frac{\partial f_\theta(x_n)}{\partial \theta}) \in \mathbb{R}^{P\times n}$ where P is the number of parameters, and define *neural tangent kernel* $K = G^T G$. Let $r = (f_\theta(x_1) - y_1; \ldots; f_\theta(x_n) - y_n)$, then $\frac{dr_i}{dt} = \frac{\partial f_\theta(x_i)}{\partial \theta}\sum_j \frac{\partial f_\theta(x_j)}{\partial \theta} r_j$, or equivalently,

$$\frac{dr}{dt} = K(t) r, \tag{27}$$

When $f_\theta(x) = \theta^T x$, the matrix $K(t)$ reduces to a constant matrix $X^T X$, thus (27) reduces to (25).

Jacot et al. [94] proved that $K(t)$ is a constant matrix for any t under certain conditions. More specifically, if the initial weights are i.i.d. Gaussian with certain variance (similar to LeCun initialization), then as the number of neurons at each layer goes to infinity sequentially, $K(t)$ converges to a constant matrix K_c (uniformly for all $t \in [0, T]$ where T is a given constant). Under further assumptions on the activations (non-polynomial activations) and data (distinct data from the unit sphere), [94] proves that K_c is positive definite. One interesting part of [94] is that

the limiting NTK matrix K_c has a closed form expression, computed recursively by an analytical formula.

Yang [232] and Novak et al. [159] extended [94]: they only require the width of each layer goes to infinitely simultaneuously (instead of sequentially in [94]), and provides a formula of NTK for convolutional networks, called CNTK.

Finite-width Ultra-wide networks. Around the same time as [94], Allen-Zhu et al. [4] and Zou et al. [255] and Du et al. [58] analyzed deep ultra-wide non-linear networks and prove that with Gaussian initialization and small enough step-size, GD and/or SGD converge to global minima (these works can be viewed extensions of an analysis of a 1-hidden-layer networks [121, 58]). In contrast to the landscape results [118, 157] that only require one layer to have n neurons, these works require a much larger number of neurons per layer: $O(n^{24}L^{12}/\delta^8)$ in [4] where $\delta = \min_{i\neq j} \|x_i - x_j\|$ and $O(n^4/\lambda_{\min}(K)^4)$ in [58] where K is a complicated matrix defined recursively. Arora et al. [10] also analyzed finite-width networks, by proving a non-asymptotic version of the NTK result of [94]. Zhang et al. [246], Ma et al. [139] analyzed the convergence of over-parameterized ResNet.

Empirical computation by NTK. The explicit formula of the limiting NTK makes it possible to actually compute NTK and perform kernel gradient descent for a real-world problem. As computing the CNTK directly is time consuming, Novak et al. [159] used Monte Carlo sampling to approximately compute CNTK. Arora et al. [10] proposed an exact efficient algorithm to compute CNTK and tests it on CIFAR10, achieving 77% test accuracy for CNTK with global average pooling. Li et al. [123] utilized two further tricks to achieve 89% test accuracy on CIFAR10, on par with AlexNet.

Mean-field approximation: another group of works. There are another group of works which also studied infinite-width limit of SGD. Sirignano and Spiliopoulos [193] considered discrete-time SGD for infinite-width multi-layer neural networks, and showed that the limit of the neural network output satisfies a certain differential equation. Arajo et al. [7], Nguyen [155] also studied infinite-width multi-layer networks. These works are extensions of previous works Mei et al. [144], Srignanao and Spiliopoulos [192] and Rotskoff and Vanden-Eijnden [177], which analyzed 1-hidden-layer networks. A major difference between these works and [94] [4] [255] [58] is the scaling factor; for instance, Sirignano and Spiliopoulos [192] considered the scaling factor 1/fan-in, while [94] [4] [255] [58] considered the scaling factor $1/\sqrt{\text{fan-in}}$. The latter scaling factor of $1/\sqrt{\text{fan-in}}$ is used in LeCun initialization (corresponding to variance 1/fan-in), thus closer to practice, but they imply that the parameters mover very little as the number of parameters increase. In contrast, [144, 192, 177, 193, 7, 155] show that the parameters evolve according to a PDE and thus can move far away from the initial point.

"Lazy training" and two learning schemes. The high-level idea of [94] [4] [255] [58] is termed "lazy training" by [41]: the model behaves like its linearization around its initial point. Because of the huge number of parameters, each parameter only needs to move a tiny amount, thus linearization is a good approximation. However, practical networks are not ultra-wide, thus the parameters will move a reasonably large amount of distance, and likely to move out of the linearization regimes. [41] indeed showed that the behavior of SGD in practical neural-nets is

different from lazy training. In addition, [41] pointed out that "lazy training" is mainly due to implicit choice of the scaling factor, and applies to a large class of models beyond neural networks. A natural question is whether the "adaptive learning scheme" described by [144, 192, 177, 193, 7, 155] can partially characterize the behavior of SGD. In an effort to answer this question, Williams et al. [221] analyzed a 1-hidden-layer ReLU network with 1-dimensional input, and provided conditions for the "kernel learning scheme" and "adaptive learning scheme".

Discussions. Math is always about simplification. Landscape analysis ignores the algorithmic aspects and focus on geometry (like geometricians). Analysis of gradient dynamics provides a more precise description of the algorithm (like dynamical systems theorists), but requires strong assumptions such as a very large width. A major difference is the point of departure. Landscape analysis only studies one aspect of the whole theory (as discussed in Section 1.1, this is common in machine learning), while algorithmic analysis aims to provide an end-to-end analysis that covers all aspects of optimization. From a theoretical perspective, it is very difficult to understand every aspect of an algorithm (even for interior point methods there are unknown questions), thus some aspects have to be ignored. The question is whether essential aspects have been captured and/or ignored. One may argue that the trajectory of the algorithm is crucial, thus landscape analysis ignores some essential part. One could also argue that moving outside of a tiny neighborhood is important, thus "lazy training" ignores some essential part. Nevertheless, from the angle of extracting some useful insight, landscape analysis has led to the discovery of mode connectivity and algorithmic analysis has led to empirical CNTK, so both have shown their potential.

6.4 Research in Shallow Networks after 2012

For the ease of presentation, results for shallow networks are mainly reviewed in this subsection. Due to the large amount of literature in GON area, it is hard to review all recent works, and we can only give an incomplete overview. We group these works based on the following criteria: landscape or algorithmic analysis (first-level classification criterion); one-neuron, 2-layer network or 1-hidden-layer network [17] (second-level criterion). Note that among the works in the same class, they may differ on the assumption on input data (Gaussian input and linearly separable input are common), number of neurons, loss function and specific algorithms (GD, SGD or others). Note that this section focuses on positive results, and negative results for shallow networks are discussed in Section 6.3.1.

Global landscape of 1-hidden-layer neural-nets. There have been many works on the landscape of 1-hidden-layer neural-nets. One interesting work (mentioned earlier when discussing mode connectivity) is Freeman and Bruna [67] which proved that the sub-level set is connected for deep linear networks and 1-hidden-layer ultra-wide ReLU networks. This does not imply every local-min is global-min, but implies there is no spurious valley (and no bad strict local-min). A

[17] In this section, we will use "2-layer network" to denote a network like $y = \phi(Wx + b)$ or $y = V^*\phi(Wx + b)$ with fixed V^*, and use "1-hidden-layer network" to denote a network like $y = V\phi(Wx + b_1) + b_2$ with both V and W being variables.

related recent work is Venturi et al. [215] which proved no spurious valley exists (implying no bad basin) for 1-hidden-layer network with "low intrinsic dimension". Haeffele and Vidal [81] extended the classical work of Burer and Monteiro [34] to 1-hidden-layer neural-net, and proved that a subset of the local minima are global minima, for a set of positive homogeneous activations. Ge et al. [70] and Gao et al. [68] designed a new loss function so that all local minima are global minima. Feizi et al. [62] designed a special network for which almost all local minima are global minima. Panigrahy et al. [166] analyzed local minima for many special neurons via electrodynamics theory. For quadratic activations, Soltanolkotabi et al. [198] proved that 2-layer over-parameterized network (with Gaussian input) have no bad local-min, and Liang et al. [125] provided a sufficient and necessary condition for the data distribution so that 1-hidden-layer neural-net has no bad local-min. For 1-hidden-layer ReLU networks (without bias term), Soudry and Hoffer [199] proved that the number of differentiable local minima is very small. Nevertheless, Laurent and von Brecht [112] showed that except flat bad local minima, all local minima of 1-hidden-layer ReLU networks (with bias term) are non-differentiable.

Algorithmic analysis of 2-layer neural-nets. There are many works on the algorithmic analysis of SGD for shallow networks under a variety of settings. The first class analyzed SGD for 2-layer neural-networks (with the second layer weights fixed). A few works mainly analyzed one single neuron. Tian [209] and Soltanolkotabi [198] analyzed the performance of GD for a single ReLU neuron. Mei et al. [144] analyzed a single sigmoid neuron. Other works analyzed 2-layer networks with multiple neurons. Brutzkus and Globerson [33] analyzed a non-overlapping 2-layer ReLU network and proved that the problem is NP-complete for general input, but if the input is Gaussian then GD converges to global minima in polynomial time. Zhong et al. [249] analyzed 2-layer under-parameterized network (no more than d neurons) for Gaussian input and initialization by tensor method. Li et al. [122] analyzed 2-layer network with skip connection for Gaussian input. Brutzkus et al. [32] analyzed 2-layer over-parameterized network with leaky ReLU activation for linearly seperable data. Wang et al. [218] and Zhang et al. [248] analyzed 2-layer over-parameterized network with ReLU activation, for linearly separable input and Gaussian input respectively. Du et al. [56] analyzed 2-layer over-parameterized network with quadratic neuron for Gaussian input. Oymak and Soltanolkotabi [165] proved the global convergence of GD with random initialization for a 2-layer network with a few types of neuron activations, when the number of parameters exceed $O(n^2)$ ($O(\cdot)$ here hides condition number and other parameters). Su and Yang [202] analyzed GD for 2-layer ReLU network with $O(n)$ neurons for generic input data.

Algorithmic analysis of 1-hidden-layer neural-nets. The second class analyzed 1-hidden-layer neural-network (with the second layer weights trainable). The relation of 1-hidden-layer network and tensors is explored in [97, 150]. Boob and Lan [27] analyzed a specially designed alternating minimization method for over-parameterized 1-hidden-layer neural-net. Du et al. [57] analyzed an non-overlapping network for Gaussian input and with an extra normalization, and proved that SGD can converge to global-min for some initialization and converge to bad local-min for other initialization. Vempala and Wilmes [214] proved that for random initialization and with $n^{O(k)}$ neurons, GD converges to the best degree k polynomial approximation of the target

function; a matching lower bound is also proved. Ge et al. [71] analyzed a new spectral method for learning 1-hidden-layer network. Oymak and Soltanolkotabi [164] analyzed GD for a rather general problem and applied it to 1-hidden-layer neural-net where $n \leq d$ (number of samples no more than dimension) for any number of neurons.

7 Concluding Remarks

In this article, we have reviewed existing theoretical results related to neural network optimization, mainly focusing on the training of feedforward neural networks. The goal of theory in general is two-fold: understanding and design. As for understanding, now we have a good understanding on the effect of initialization on stable training, and some understanding on the effect of over-parameterization on the landscape. As for design, theory has already greatly helped the design of algorithms (e.g. initialization schemes, batch normalization, Adam). There are also examples like CNTK that is motivated from theoretical analysis and has become a real tool. Besides design and understanding, some interesting empirical phenomenons have been discovered, such as mode connectivity and lottery ticket hypothesis, awaiting more theoretical studies. Overall, there is quite some progress in the theory for neural-net optimization.

That being said, there are still lots of challenges. As for understanding, we still do not understand many of the components that affect the performance, e.g., the detailed architecture and Adam optimizer. With the current theory, it is still far from making a good prediction on the performance of an algorithm, especially in a setting that is different from the classification problem. As for design, one major challenge for the theoretical researchers is that the chance of (strong) theoretically-driven algorithms for image classification seems low. Opportunities may lie in other areas, such as robustness to adversarial attacks and deep reinforcement learning.

8 Acknowledgement

We would like to thank Leon Bottou, Yan Dauphin, Yuandong Tian, Levent Sagun, Lechao Xiao, Tengyu Ma, Jason Lee, Matus Telgarsky, Ju Sun, Wei Hu, Simon Du, Lei Wu, Quanquan Gu, Justin Sirignano, Shiyu Liang, R. Srikant, Tian Ding and Dawei Li for discussions on various results reviewed in this article. We also thank Ju Sun for the list of related works in the webpage [101] which helps the writing of this article.

A Discussion of General Convergence Result

This section is an extension of Section 3.2 on the convergence analysis.

Convergence of iterates. Recall that the statement "every limit point is a stationary point" does not eliminate two undesirable cases: (U1) the sequence could have more than one limit points; (U2) limit points could be non-existent.

It is relatively easy to eliminate the possibility of (U1) since for most neural network training problem, Kurdyka-Lojasiewicz condition holds (see, e.g. [243]), and together with some minor conditions, it is not hard to show that there can be no more than one limit points for a descent algorithm (see, e.g., [138, 12]). Nevertheless, a rigorous argument for a generic neural network optimization problem is not easy.

Eliminating the possibility of (U2) is both easy and hard. It is easy in the sense that the divergence is often excluded by adding extra assumptions such as compactness of level sets $\{x \mid f(x) \leq c\}$, which can be enforced by adding a proper regularizer. It is hard since for neural-net optimization, the required regularizer may be impractical (e.g. a degree $2L$ polynomial for quadratic loss). Another solution is to add a ball constraint on the variables, but that will cause other issues (e.g. convergence analysis of SGD for constrained problems is complicated). Thus, if one really wants to eliminate (U2), a tailored analysis for neural-nets may be needed.

Global Lipschitz constants. Global Lipschitz smoothness of gradient is required for GD to converge, but neural network optimization problems do not have a global Lipschitz constant. An intuitive solution is to use a local Lipschitz constant for picking the stepsize instead of a global Lipschitz constant, but it seems hard to provide a clean result. We discuss a few plausible solutions and mention their issues.

A natural choice is to use stepsize dependent on local Lipschitz constant (i.e. the largest eigenvalue of the Hessian at the current iterate). To prove the convergence, we can modify the proof of Proposition 1.2.3 in [24], but the proof does not work directly since the step-size could go to zero too fast, and does not satisfy the condition $\sum_t \eta_t = \infty$. One may wonder whether GD with stepsize dependent on local Lipschitz constant is a special case of the "successive upper-bound minimization" framework of [175] and the convergence theorem such as Theorem 3 in [175] can directly apply. However, it is not a special case of [175] since that paper requires a "global upper bound" of the objective function, but using a local Lipschitz constant only provides a local upper bound.

Another idea is to utilize the fact that the gradient is Lipschitz smooth in a compact set such as a ball. For instance, for matrix factorization problems, using local Lipschitz constant in a ball is a common approach, e.g., Lemma 1 of [40]. However, this lemma also requires convexity in the ball to ensure the iterates do not move out of the ball. For general non-convex functions, it is not easy to prove that the algorithm does not move out of the ball. As an exercise, readers can try to analyze $\min_w (1 - w^6)^2$, and see whether it is easy to prove GD with stepsize chosen based on the

Lipschitz constant in a ball converges [18].

To remedy the idea of using Lipschitz constant of a compact set, a natural solution is to keep the iterates bounded. But how to ensure the iterates are bounded? One way is to add constraints on the variables, but this may cause other difficulties since it becomes a constrained problem. Another solution is to add regularizers such as $\|\theta\|^2$, $\max\{\|\theta\|^2, B\}$, or a smooth version $(\max\{\|\theta\|^2 - B, 0\})^2$ as used in [203], where B is large enough such that at least one global minimum has norm no more than $\sqrt{B}$. However, it seems not easy to rigorously prove convergence even with the aid of regularizers. In addition, the existing landscape analysis or global convergence are done for non-regularized problems, and new analysis is required if regularizers are added. Again, analysis tailored for a specific problem may prove these, but for now we are hoping for a clean universal analysis. In short, a simple modification to the problem seems hard to ensure the existence of a constant stepsize that guarantees the convergence [19].

For most practitioners, there is a simple conceptual solution to understand convergence: adding a "posterior" assumption on the boundedness of iterates. More specifically, assuming the iterates are bounded, then GD with a proper constant step-size converges in the sense that the gradient converges to 0. The assumption itself can be verified in practice, but it is only verifiable after running the algorithm (thus a "posterior" assumption). Anyhow, this is one of the many imperfections of the current theory, and we have to put it aside and move on to other aspects of the problem.

B Details of Batch Normalization

First trick of BatchNorm. There are two extra tricks in the implemented BatchNorm, and the first trick is to add two more parameters to restore the representation power. We define a formal BN operation as follows. For scalars $a_1, a_2, \ldots, a_N$, define $\mu = \frac{1}{N}(a_1 + \cdots + a_N)$ and $\sigma = \sqrt{\frac{1}{N}(a_i - \mu)^2}$, then

$$\mathrm{BN}_{\gamma,\beta}(a_1, \ldots, a_N) \triangleq \left(\gamma \frac{a_1 - \mu}{\sigma + \epsilon} + \beta, \ldots, \gamma \frac{a_N - \mu}{\sigma + \epsilon} + \beta\right),$$

where $\gamma \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$ are parameters to be learned, and ϵ is a fixed small constant. This BN operation is a mapping from input $a_1, \ldots, a_N$ to output $\mathrm{BN}_{\gamma,\beta}(a_1, \ldots, a_N)$ and is differentiable. BN operation is defined as a general function applicable to any N scalars, and we discuss how to incorporate it into the neural network next. In a neural network, this BN operation is added before each nonlinear transformation layer in the neural network f_θ to obtain a new neural network $\tilde{f}_{\tilde{\theta}}$, where $\tilde{\theta}$ involves the new parameters $\{\gamma^l, \beta^l\}_{l=1}^L$.

We illustrate by a 1-dimensional 2-layer neural network. Suppose the input instances are $x_1, \ldots, x_n \in \mathbb{R}$, and the original neural network is $\hat{y}_i = v\phi(wx_i), i = 1, \ldots, n$ where $v, w \in \mathbb{R}$. The new network with BN operations is $(\hat{y}_1, \ldots, \hat{y}_n) = v\phi(\mathrm{BN}_{\gamma,\beta}(wx_1, \ldots, wx_n))$. The problem

[18] We do not know a clean proof that is generalizable to high-dimensional problems. All our current proofs utilize certain property of the problem which highly relies on the 1-dimensional structure, and are thus not that interesting.

[19] In fact, even dealing with convex problems without Lipschitz constant is not an easy problem in optimization, and only until recently are there good progress in some convex problems [20, 135].

formulation has also been changed: previously, the objective function can be decomposed across samples $F(\theta) = \sum_{i=1}^{n} \ell(y_i, f_\theta(x_i))$, now the objective function cannot be decomposed. In this 1-dimensional example, there is only one feature at each layer. For a high-dimensional neural network, the BN operation applies to each feature of the pre-activations separately, and aggregates information across samples to compute the mean and variance.

Second trick of BatchNorm. In practice, the network cannot take all samples to compute the mean and variance, thus it is natural to take samples in one mini-batch (say, 64 samples) to compute the mean and variance. Suppose there are N samples in a mini-batch, then the new network takes N inputs jointly and produce N predictions, i.e., $(\hat{y}_1, \ldots, \hat{y}_N) = \tilde{f}_{\tilde{\theta}}(x_1, \ldots, x_N)$. Now the objective function can be decomposed across mini-batches. Mini-batch stochastic gradient descent (or other methods) can still be applied to the new objective function to learn the weights [20].

Inference stage. Finally, there is a small trick for the inference stage. After training the network, one needs to perform inference for new data (e.g. predict the labels of unseen images). If we rigorously follow the paradigm of training/test, then we need to take a mini-batch of test samples as input to the network. Nevertheless, in practice one often uses the mean and variance computed for the training data, and thus the network can make prediction for each single test sample.

C Theoretical Complexity of Large-scale Optimization Methods

In this section, we review the theoretical complexity of a few optimization methods for large-scale optimization. We explain the explicit convergence rate and computational complexity, in order to reveal the connection and differences of various methods.

To unify these methods in one framework, we start with the common convergence rate results of gradient descent method (GD) and explain how different methods improve the convergence rate in different ways. Consider the prototype convergence rate result in convex optimization: the epoch-complexity [21] is $O(\kappa \log 1/\epsilon)$ or $O(\beta/\epsilon)$. These rates mean the following: to achieve ϵ error, the number of epochs to achieve error ϵ is no more than $\kappa \log 1/\epsilon$ for strongly convex problems (or β/ϵ for convex problems), where κ is the condition number of the problem (and β is the global Lipschitz constant of the gradient). For simplicity, we focus on strongly convex problems.

There are at least four classes of methods that can improve the computation time of GD for strongly convex problems [22].

The first class of methods are parallel computation methods. This method mainly saves the

[20] See also Section 8.7.1 of [77] for a description.

[21] For batch GD, one epoch is one iteration. For SGD, one epoch consists of multiple stochastic gradient steps that pass all data points once. We do not say "iteration complexity" or "the number of iterations" since per-iteration cost for the vanilla gradient descent and SGD are different and can easily cause confusion. In contrast, the per-epoch cost (number of operations) for batch GD and SGD are comparable.

[22] Note that the methods discussed below also improve the rate for convex problems but we skip the discussions on convex problems.

per-epoch computation time, instead of improving the overall convergence speed. For example, for minimizing an n-dimensional least square problem, each epoch of GD requires a matrix-vector product which is parallelizable. More specifically, while a serial implementation takes time $O(n^2)$ to perform a matrix-vector product, a parallel model can take time as little as $O(\log n)$. This is a simplified discussion, and many other factors such as the computation graph of the hardware, synchronization cost and the communication cost can greatly affect the performance. In general, parallel computation is quite complicated, which is why an area called parallel computation is devoted to this topic (see the classical book [25] for an excellent discussion of the intersection of parallel computation and numerical optimization). For deep learning, as discussed earlier, using K machines to achieve nearly K-times speedup has been a popular thread of research.

The second class of methods are fast gradient methods (FGM) that have convergence rate $O(\sqrt{\kappa}\log 1/\epsilon)$, thus saving a factor of $\sqrt{\kappa}$ compared to the convergence rate of GD $O(\kappa \log 1/\epsilon)$. FGM includes conjugate gradient method, heavy ball method and accelerated gradient method. For convex quadratic problems, these three methods all achieve the improved rate $O(\sqrt{\kappa}\log 1/\epsilon)$. For general strongly convex problems, only accelerated gradient method is known to achieve the rate $O(\sqrt{\kappa}\log 1/\epsilon)$.

The third class of methods are based on decomposition, i.e. decomposing a large problem into smaller ones. Due to the hardware limit and the huge number of data/parameters, it is often impossible to process all samples/parameters at once, thus loading data/parameters separately becomes a necessity. In this serial computation model, GD can still be implemented (e.g. by gradient accumulation), but it is not the fastest method. Better methods are decomposition-based methods, including SGD, coordinate descent (CD) and their mixture. To illustrate their theoretical benefits, consider an unconstrained d-dimensional least squares problem with n samples. For simplicity, assume $n \geq d$ and the Hessian matrix has rank d.

- Randomized CD has an epoch-complexity $O(\kappa_{\mathrm{D}} \log 1/\epsilon)$ [117, 152], where κ_{D} is the ratio of the average eigenvalue λ_{avg} over the minimum eigenvalue $\lambda_{\min}$ of the coefficient matrix, and is related to Demmel's condition number. This is smaller than the rate of GD $O(\kappa \log 1/\epsilon)$ by a factor of $\lambda_{\max}/\lambda_{\mathrm{avg}}$ where $\lambda_{\max}$ is the maximum eigenvalue. Clearly, the improvement ratio $\lambda_{\max}/\lambda_{\mathrm{avg}}$ lies in $[1, d]$, thus randomized CD is 1 to d times faster than GD.

- Recent variants of SGD (such as SVRG [100] and SAGA [48]) achieve an epoch-complexity $O(\frac{n}{d}\kappa_{\mathrm{D}} \log 1/\epsilon)$, which is 1 to n times faster than GD. When $n = d$, this complexity is the same as R-CD for least squares problems (though not pointed out in the literature). We highlight that this up-to-n-factor acceleration has been the major focus of recent studies of SGD type methods, and has achieved much attention in theoretical machine learning area.

- Classical theory of vanilla SGD [30] often uses diminishing stepsize and thus does not enjoy the same benefit as SVRG and SAGA. However, as discussed earlier, for realizable least squares problem, SGD with constant step-size can achieve an epoch-complexity $O(\frac{n}{d}\kappa_{\mathrm{D}} \log 1/\epsilon)$, which is 1 to n times faster than GD.

The above discussions are mainly for single sample/coordinate algorithms. If the samples/coordinates are grouped in mini-batches/blocks, the maximal acceleration ratio is roughly the number of mini-batches/blocks.

The fourth class of methods utilize the second order information of the problem, including quasi-Newton method and GD with adaptive learning rates. Quasi-Newton methods such as BFGS and limited BFGS (see, e.g., [223]) use an approximation of the Hessian in each epoch, and are popular choices for many nonlinear optimization problems. AdaGrad, RMSProp, Adam and Barzilai-Borwein (BB) method use a diagonal matrix estimation of the Hessian. It seems very difficult to theoretically justify the advantage of these methods over GD, but intuitively, the convergence speed of these methods rely much less on the condition number κ (or any variant of the condition number such as κ_{D}).

We briefly summarize the benefits of these methods as below. Consider minimizing d-dimensional least squares problem with $n = d$ samples, and suppose each machine can process at most one sample or one coordinate at once, which takes one time unit. The benchmark GD takes time $O(n\kappa \log 1/\epsilon)$ to achieve error ϵ. With n machines, the first class (parallel computation) can potentially reduce the computation time to $O(\kappa \log n \log 1/\epsilon)$ with extra cost such as communication. For other methods, we assume only one machine is available. The second class (e.g. accelerated gradient method) reduces the computation time to $O(n\sqrt{\kappa} \log 1/\epsilon)$, the third class (e.g. SVRG and R-CD) reduces the computation time to $O(n\kappa_{\mathrm{D}} \log 1/\epsilon)$, and the fourth class (e.g. BFGS and BB) may improve κ to other parameters that are unclear. Although we treat these methods as separate classes, researchers have extensively studied various mixed methods of two or more classes, though the theoretical analysis can be much harder.

References

[1] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[2] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *The Journal of Machine Learning Research*, 18(1):8194–8244, 2017.

[3] Zeyuan Allen-Zhu. Natasha 2: Faster non-convex optimization than sgd. In *Advances in Neural Information Processing Systems*, pages 2680–2691, 2018.

[4] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*, 2018.

[5] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2007.

[6] Shun-Ichi Amari, Hyeyoung Park, and Kenji Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12(6):1399–1409, 2000.

[7] Dyego Arajo, Roberto I. Oliveira, and Daniel Yukimura. A mean-field limit for certain deep neural networks, 2019.

[8] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[9] Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018.

[10] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.

[11] Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rkxQ-nA9FX`.

[12] Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods. *Mathematical Programming*, 137(1-2):91–129, 2013.

[13] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[14] Marco Baity-Jesi, Levent Sagun, Mario Geiger, Stefano Spigler, G Ben Arous, Chiara Cammarota, Yann LeCun, Matthieu Wyart, and Giulio Biroli. Comparing dynamics: Deep neural networks versus glassy systems. *arXiv preprint arXiv:1803.06969*, 2018.

[15] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 342–350. JMLR. org, 2017.

[16] Anas Barakat and Pascal Bianchi. Convergence analysis of a momentum algorithm with adaptive step size for non convex optimization. *arXiv preprint arXiv:1911.07596*, 2019.

[17] Peter Bartlett, Dave Helmbold, and Phil Long. Gradient descent with identity initialization efficiently learns positive definite linear transformations. In *International Conference on Machine Learning*, pages 520–529, 2018.

[18] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

[19] Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

[20] Heinz H Bauschke, Jérôme Bolte, and Marc Teboulle. A descent lemma beyond lipschitz gradient continuity: first-order methods revisited and applications. *Mathematics of Operations Research*, 42(2): 330–348, 2016.

[21] Sue Becker, Yann Le Cun, et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

[22] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning and the bias-variance trade-off. *arXiv preprint arXiv:1812.11118*, 2018.

[23] Albert S Berahas, Majid Jahani, and Martin Takáč. Quasi-newton methods for deep learning: Forget the past, just sample. *arXiv preprint arXiv:1901.09997*, 2019.

[24] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3): 334–334, 1997.

[25] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[26] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. In *Advances in Neural Information Processing Systems*, pages 7694–7705, 2018.

[27] Digvijay Boob and Guanghui Lan. Theoretical properties of the global optimizer of two layer neural network. *arXiv preprint arXiv:1710.11241*, 2017.

[28] Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10(Jul):1737–1754, 2009.

[29] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168, 2008.

[30] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[31] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[32] A. Brutzkus, A. Globerson, E. Malach, and S. Shalev-Shwartz. Sgd learns over-parameterized networks that provably generalize on linearly separable data. *ICLR*, 2018.

[33] Alon Brutzkus and Amir Globerson. Globally optimal gradient descent for a convnet with gaussian inputs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 605–614. JMLR. org, 2017.

[34] Samuel Burer and Renato DC Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444, 2005.

[35] Yongqiang Cai, Qianxiao Li, and Zuowei Shen. A quantitative analysis of the effect of batch normalization on gradient descent. In *International Conference on Machine Learning*, pages 882–890, 2019.

[36] Yair Carmon, John C Duchi, Oliver Hinder, and Aaron Sidford. Convex until proven guilty: Dimension-free acceleration of gradient descent on non-convex functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 654–663. JMLR. org, 2017.

[37] Yair Carmon, John C Duchi, Oliver Hinder, and Aaron Sidford. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018.

[38] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

[39] Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*, 2018.

[40] Yuejie Chi, Yue M Lu, and Yuxin Chen. Nonconvex optimization meets low-rank matrix factorization: An overview. *IEEE Transactions on Signal Processing*, 67(20):5239–5269, 2019.

[41] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming, 2018.

[42] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[43] Frank E Curtis and Katya Scheinberg. Optimization methods for supervised machine learning: From linear models to deep learning. In *Leading Developments from INFORMS Communities*, pages 89–114. INFORMS, 2017.

[44] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[45] Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence guarantees for rmsprop and adam in non-convex optimization and an empirical comparison to nesterov acceleration. 2018.

[46] Aaron Defazio and Léon Bottou. On the ineffectiveness of variance reduced optimization for deep learning. In *Advances in Neural Information Processing Systems*, pages 1753–1763, 2019.

[47] Aaron Defazio and Léon Bottou. Scaling laws for the principled design, initialization and preconditioning of relu networks. *arXiv preprint arXiv:1906.04267*, 2019.

[48] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[49] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[50] Olivier Devolder, François Glineur, Yurii Nesterov, et al. First-order methods with inexact oracle: the strongly convex case. *CORE Discussion Papers*, 2013016, 2013.

[51] Olivier Devolder, François Glineur, and Yurii Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Mathematical Programming*, 146(1-2):37–75, 2014.

[52] Tian Ding, Dawei Li, and Ruoyu Sun. Spurious local minima exist for almost all over-parameterized neural networks. *optimization online*, 2019.

[53] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1019–1028. JMLR. org, 2017.

[54] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[55] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A Hamprecht. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

[56] Simon S Du and Jason D Lee. On the power of over-parametrization in neural networks with quadratic activation. *arXiv preprint arXiv:1803.01206*, 2018.

[57] Simon S Du, Jason D Lee, Yuandong Tian, Barnabas Poczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. *arXiv preprint arXiv:1712.00779*, 2017.

[58] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

[59] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[60] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[61] Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems*, pages 687–697, 2018.

[62] Soheil Feizi, Hamid Javadi, Jesse Zhang, and David Tse. Porcupine neural networks:(almost) all local optima are global. *arXiv preprint arXiv:1710.02196*, 2017.

[63] O. P. Ferreira and S. Z. Németh. On the spherical convexity of quadratic functions. *Journal of Global Optimization*, 73(3):537–545, Mar 2019. ISSN 1573-2916. doi: 10.1007/s10898-018-0710-6. URL `https://doi.org/10.1007/s10898-018-0710-6`.

[64] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[65] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

[66] Silvio Franz, Sungmin Hwang, and Pierfrancesco Urbani. Jamming in multilayer supervised learning models. *arXiv preprint arXiv:1809.09945*, 2018.

[67] C Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*, 2016.

[68] Weihao Gao, Ashok Vardhan Makkuva, Sewoong Oh, and Pramod Viswanath. Learning one-hidden-layer neural networks under general input distributions. *arXiv preprint arXiv:1810.04133*, 2018.

[69] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.

[70] Rong Ge, Jason D Lee, and Tengyu Ma. Learning one-hidden-layer neural networks with landscape design. *arXiv preprint arXiv:1711.00501*, 2017.

[71] Rong Ge, Rohith Kuditipudi, Zhize Li, and Xiang Wang. Learning two-layer neural networks with symmetric inputs. *arXiv preprint arXiv:1810.06793*, 2018.

[72] Mario Geiger, Stefano Spigler, Stéphane d'Ascoli, Levent Sagun, Marco Baity-Jesi, Giulio Biroli, and Matthieu Wyart. The jamming transition as a paradigm to understand the loss landscape of deep neural networks. *arXiv preprint arXiv:1809.09349*, 2018.

[73] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. *arXiv preprint arXiv:1901.10159*, 2019.

[74] Dar Gilboa, Bo Chang, Minmin Chen, Greg Yang, Samuel S Schoenholz, Ed H Chi, and Jeffrey Pennington. Dynamical isometry and a mean field theory of lstms and grus. *arXiv preprint arXiv:1901.08987*, 2019.

[75] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[76] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[77] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[78] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[79] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r14EOsCqKX`.

[80] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[81] Benjamin D Haeffele and René Vidal. Global optimality in neural network training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7331–7339, 2017.

[82] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[83] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *Advances in Neural Information Processing Systems*, pages 580–589, 2018.

[84] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, pages 569–579, 2018.

[85] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

[86] Haowei He, Gao Huang, and Yang Yuan. Asymmetric valleys: Beyond sharp and flat local minima. *arXiv preprint arXiv:1902.00744*, 2019.

[87] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[89] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[90] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[91] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[92] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[93] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[94] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[95] Prateek Jain, Sham M Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent. *arXiv preprint arXiv:1704.08227*, 2017.

[96] Daniel Jakubovitz, Raja Giryes, and Miguel RD Rodrigues. Generalization error in deep learning. In *Compressed Sensing and Its Applications*, pages 153–193. Springer, 2019.

[97] Majid Janzamin, Hanie Sedghi, and Anima Anandkumar. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv preprint arXiv:1506.08473*, 2015.

[98] Ziwei Ji and Matus Telgarsky. Gradient descent aligns the layers of deep linear networks. *arXiv preprint arXiv:1810.02032*, 2018.

[99] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[100] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[101] Sun Ju. List of works on "provable nonconvex methods/algorithms". *https://sunju.org/research/nonconvex/*.

[102] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

[103] Kenji Kawaguchi and Leslie Pack Kaelbling. Elimination of all bad local minima in deep learning. *arXiv preprint arXiv:1901.00279*, 2019.

[104] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

[105] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[106] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9. IEEE, 2018.

[107] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[108] Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 806–815, 2019.

[109] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[110] Rohith Kuditipudi, Xiang Wang, Holden Lee, Yi Zhang, Zhiyuan Li, Wei Hu, Sanjeev Arora, and Rong Ge. Explaining landscape connectivity of low-cost solutions for multilayer nets. *arXiv preprint arXiv:1906.06247*, 2019.

[111] Thomas Laurent and James Brecht. Deep linear networks with arbitrary loss: All local minima are global. In *International Conference on Machine Learning*, pages 2908–2913, 2018.

[112] Thomas Laurent and James von Brecht. The multilinear structure of relu networks. *arXiv preprint arXiv:1712.10132*, 2017.

[113] Yann LeCun, Leon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

[114] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[115] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[116] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1VZqjAcYX`.

[117] Dennis Leventhal and Adrian S Lewis. Randomized methods for linear constraints: convergence rates and conditioning. *Mathematics of Operations Research*, 35(3):641–654, 2010.

[118] Dawei Li, Tian Ding, and Ruoyu Sun. Over-parameterized deep neural networks have no strict local minima for any continuous activations. *arXiv preprint arXiv:1812.11039*, 2018.

[119] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6391–6401, 2018.

[120] Ping Li and Phan-Minh Nguyen. On random deep weight-tied autoencoders: Exact asymptotic analysis, phase transitions, and implications to training. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HJx54i05tX`.

[121] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, pages 8168–8177, 2018.

[122] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems*, pages 597–607, 2017.

[123] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S. Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels, 2019.

[124] Shiyu Liang, Ruoyu Sun, Jason D Lee, and R Srikant. Adding one neuron can eliminate all bad local minima. In *Advances in Neural Information Processing Systems*, pages 4355–4365, 2018.

[125] Shiyu Liang, Ruoyu Sun, Yixuan Li, and Rayadurgam Srikant. Understanding the loss surface of neural networks for binary classification. *arXiv preprint arXiv:1803.00909*, 2018.

[126] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[127] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, pages 3384–3392, 2015.

[128] Chaoyue Liu and Mikhail Belkin. Accelerating sgd with momentum for over-parameterized learning, 2018.

[129] Chaoyue Liu and Mikhail Belkin. Mass: an accelerated stochastic method for over-parametrized learning. *arXiv preprint arXiv:1810.13395*, 2018.

[130] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[131] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

[132] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[133] Cheng Lu, Zhibin Deng, Jing Zhou, and Xiaoling Guo. A sensitive-eigenvector based global algorithm for quadratically constrained quadratic programming. *Journal of Global Optimization*, pages 1–18, 2019.

[134] Haihao Lu and Kenji Kawaguchi. Depth creates no bad local minima. *arXiv preprint arXiv:1702.08580*, 2017.

[135] Haihao Lu, Robert M Freund, and Yurii Nesterov. Relatively smooth convex optimization by first-order methods, and applications. *SIAM Journal on Optimization*, 28(1):333–354, 2018.

[136] Ping Luo, Ruimao Zhang, Jiamin Ren, Zhanglin Peng, and Jingyu Li. Switchable normalization for learning-to-normalize deep representation. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[137] Zhi-Quan Luo. On the convergence of the lms algorithm with adaptive learning rate for linear feed-forward networks. *Neural Computation*, 3(2):226–245, 1991.

[138] Zhi-Quan Luo and Paul Tseng. Error bounds and convergence analysis of feasible descent methods: a general approach. *Annals of Operations Research*, 46(1):157–178, 1993.

[139] Chao Ma, Lei Wu, et al. Analysis of the gradient descent algorithm for a deep neural network model with skip-connections. *arXiv preprint arXiv:1904.05263*, 2019.

[140] James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[141] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[142] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[143] Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and double descent curve. *arXiv preprint arXiv:1908.05355*, 2019.

[144] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layers neural networks. *arXiv preprint arXiv:1804.06561*, 2018.

[145] Hiroaki Mikami, Hisahiro Suganuma, Yoshiki Tanaka, Yuichi Kageyama, et al. Massively distributed sgd: Imagenet/resnet-50 training in a flash. *arXiv preprint arXiv:1811.05233*, 2018.

[146] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[147] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[148] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[149] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[150] Marco Mondelli and Andrea Montanari. On the connection between learning two-layers neural networks and tensor decomposition. *arXiv preprint arXiv:1802.07301*, 2018.

[151] Ari S Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *arXiv preprint arXiv:1906.02773*, 2019.

[152] Y. Nesterov. Efficiency of coordiate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[153] Behnam Neyshabur, Russ R Salakhutdinov, and Nati Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

[154] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[155] Phan-Minh Nguyen. Mean field limit of the learning dynamics of multilayer neural networks. *arXiv preprint arXiv:1902.02880*, 2019.

[156] Quynh Nguyen. On connected sublevel sets in deep learning. *arXiv preprint arXiv:1901.07417*, 2019.

[157] Quynh Nguyen, Mahesh Chandra Mukkamala, and Matthias Hein. On the loss landscape of a class of deep neural networks with no bad local valleys. *arXiv preprint arXiv:1809.10749*, 2018.

[158] Maher Nouiehed and Meisam Razaviyayn. Learning deep models: Critical points and local openness. *arXiv preprint arXiv:1803.02968*, 2018.

[159] Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1g30j0qF7`.

[160] Brendan O?donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015.

[161] Francesco Orabona and Tatiana Tommasi. Training deep networks without learning rates through coin betting. In *Advances in Neural Information Processing Systems*, pages 2160–2170, 2017.

[162] A Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

[163] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *arXiv preprint arXiv:1811.12019*, 2018.

[164] Samet Oymak and Mahdi Soltanolkotabi. Overparameterized nonlinear learning: Gradient descent takes the shortest path? *arXiv preprint arXiv:1812.10004*, 2018.

[165] Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. *arXiv preprint arXiv:1902.04674*, 2019.

[166] Rina Panigrahy, Ali Rahimi, Sushant Sachdeva, and Qiuyi Zhang. Convergence results for neural networks via electrodynamics. *arXiv preprint arXiv:1702.00458*, 2017.

[167] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[168] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[169] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In *Advances in neural information processing systems*, pages 4785–4795, 2017.

[170] Jeffrey Pennington, Samuel S Schoenholz, and Surya Ganguli. The emergence of spectral universality in deep networks. *arXiv preprint arXiv:1802.09979*, 2018.

[171] Tomaso Poggio and Qianli Liao. *Theory II: Landscape of the empirical risk in deep learning*. PhD thesis, Center for Brains, Minds and Machines (CBMM), arXiv, 2017.

[172] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

[173] Michael James David Powell. Restart procedures for the conjugate gradient method. *Mathematical programming*, 12(1):241–254, 1977.

[174] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[175] Meisam Razaviyayn, Mingyi Hong, and Zhi-Quan Luo. A unified convergence analysis of block successive minimization methods for nonsmooth optimization. *SIAM Journal on Optimization*, 23(2): 1126–1153, 2013.

[176] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018.

[177] Grant M Rotskoff and Eric Vanden-Eijnden. Neural networks as interacting particle systems: Asymptotic convexity of the loss landscape and universal scaling of the approximation error. *arXiv preprint arXiv:1805.00915*, 2018.

[178] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[179] Itay Safran and Ohad Shamir. Spurious local minima are common in two-layer relu neural networks. *arXiv preprint arXiv:1712.08968*, 2017.

[180] Levent Sagun, Léon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*, 2016.

[181] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[182] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[183] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[184] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[185] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pages 343–351, 2013.

[186] Mark Schmidt and Nicolas Le Roux. Fast convergence of stochastic gradient descent under a strong growth condition. *arXiv preprint arXiv:1308.6370*, 2013.

[187] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[188] Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJevYoA9Fm`.

[189] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3067–3075. JMLR. org, 2017.

[190] Ohad Shamir. Exponential convergence time of gradient descent for one-dimensional deep linear neural networks. *arXiv preprint arXiv:1809.08587*, 2018.

[191] Yeonjong Shin. Effects of depth, width, and initialization: A convergence analysis of layer-wise training for deep linear neural networks, 2019.

[192] Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks. *arXiv preprint arXiv:1805.01053*, 2018.

[193] Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of deep neural networks, 2019.

[194] Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and Franois Fleuret. On the tunability of optimizers in deep learning, 2019.

[195] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[196] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. *arXiv preprint arXiv:1708.07120*, 2017.

[197] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1Yy1BxCZ`.

[198] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65(2):742–769, 2019.

[199] Daniel Soudry and Elad Hoffer. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777*, 2017.

[200] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012.

[201] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[202] Lili Su and Pengkun Yang. On learning over-parameterized neural networks: A functional approximation prospective. In *Advances in Neural Information Processing Systems*.

[203] Ruoyu Sun and Zhi-Quan Luo. Guaranteed matrix completion via non-convex factorization. *IEEE Transactions on Information Theory*, 62(11):6535–6579, 2016.

[204] Grzegorz Swirszcz, Wojciech Marian Czarnecki, and Razvan Pascanu. Local minima in training of deep networks. 2016.

[205] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[206] Conghui Tan, Shiqian Ma, Yu-Hong Dai, and Yuqiu Qian. Barzilai-borwein step size for stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 685–693, 2016.

[207] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[208] Wojciech Tarnowski, Piotr Warchoł, Stanisław Jastrz?bski, Jacek Tabor, and Maciej Nowak. Dynamical isometry is achieved in residual networks in a universal way for any activation function. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2221–2230, 2019.

[209] Yuandong Tian. An analytical formula of population gradient for two-layered relu network and its applications in convergence and critical point analysis. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3404–3413. JMLR. org, 2017.

[210] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[211] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[212] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[213] Sharan Vaswani, Francis Bach, and Mark Schmidt. Fast and faster convergence of sgd for over-parameterized models and an accelerated perceptron. *arXiv preprint arXiv:1810.07288*, 2018.

[214] Santosh Vempala and John Wilmes. Polynomial convergence of gradient descent for training one-hidden-layer neural networks. *arXiv preprint arXiv:1805.02677*, 2018.

[215] Luca Venturi, Afonso Bandeira, and Joan Bruna. Neural networks with finite intrinsic dimension have no spurious valleys. *arXiv preprint arXiv:1802.06384*, 15, 2018.

[216] Luca Venturi, Afonso Bandeira, and Joan Bruna. Spurious valleys in two-layer neural network optimization landscapes. *arXiv preprint arXiv:1802.06384*, 2018.

[217] Rene Vidal, Joan Bruna, Raja Giryes, and Stefano Soatto. Mathematics of deep learning. *arXiv preprint arXiv:1712.04741*, 2017.

[218] Gang Wang, Georgios B Giannakis, and Jie Chen. Learning relu networks on linearly separable data: Algorithm, optimality, and generalization. *arXiv preprint arXiv:1808.04685*, 2018.

[219] Rachel Ward, Xiaoxia Wu, and Leon Bottou. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, 2018.

[220] Colin Wei, Jason D Lee, Qiang Liu, and Tengyu Ma. On the margin theory of feedforward neural networks. *arXiv preprint arXiv:1810.05369*, 2018.

[221] Francis Williams, Matthew Trager, Claudio Silva, Daniele Panozzo, Denis Zorin, and Joan Bruna. Gradient dynamics of shallow univariate relu networks, 2019.

[222] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[223] Stephen Wright and Jorge Nocedal. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[224] Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

[225] Yuxin Wu and Kaiming He. Group normalization. *ECCV*, 2018.

[226] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. *arXiv preprint arXiv:1806.05393*, 2018.

[227] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[228] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[229] Yi Xu, Jing Rong, and Tianbao Yang. First-order stochastic algorithms for escaping from saddle points in almost linear time. In *Advances in Neural Information Processing Systems*, pages 5535–5545, 2018.

[230] Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima. Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds. *arXiv preprint arXiv:1903.12650*, 2019.

[231] Ge Yang and Samuel Schoenholz. Mean field residual networks: On the edge of chaos. In *Advances in neural information processing systems*, pages 7103–7114, 2017.

[232] Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

[233] Mingyang Yi, Qi Meng, Wei Chen, Zhi-ming Ma, and Tie-Yan Liu. Positively scale-invariant flatness of relu neural networks. *arXiv preprint arXiv:1903.02237*, 2019.

[234] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018.

[235] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, page 1. ACM, 2018.

[236] Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. Reducing bert pre-training time from 3 days to 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[237] Jiahui Yu and Thomas Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

[238] X. Yu and S. Pasupathy. Innovations-based MLSE for Rayleigh flat fading channels. *IEEE Transacations on Communications*, pages 1534–1544, 1995.

[239] Ya-xiang Yuan. Step-sizes for the gradient method. *AMS IP Studies in Advanced Mathematics*, 42(2): 785, 2008.

[240] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Global optimality conditions for deep neural networks. *arXiv preprint arXiv:1707.02444*, 2017.

[241] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Small nonlinearities in activation functions create bad local minima in neural networks. *arXiv preprint arXiv:1802.03487*, 2018.

[242] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[243] Jinshan Zeng, Tim Tsz-Kit Lau, Shaobo Lin, and Yuan Yao. Block coordinate descent for deep learning: Unified convergence guarantees. *arXiv preprint arXiv:1803.00225*, 2018.

[244] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[245] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

[246] Huishuai Zhang, Da Yu, Wei Chen, and Tie-Yan Liu. Training over-parameterized deep resnet is almost as easy as training a two-layer network. *arXiv preprint arXiv:1903.07120*, 2019.

[247] Li Zhang. Depth creates no more spurious local minima. *arXiv preprint arXiv:1901.09827*, 2019.

[248] Xiao Zhang, Yaodong Yu, Lingxiao Wang, and Quanquan Gu. Learning one-hidden-layer relu networks via gradient descent. *arXiv preprint arXiv:1806.07808*, 2018.

[249] Kai Zhong, Zhao Song, Prateek Jain, Peter L Bartlett, and Inderjit S Dhillon. Recovery guarantees for one-hidden-layer neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4140–4149. JMLR. org, 2017.

[250] Dongruo Zhou, Yiqi Tang, Ziyan Yang, Yuan Cao, and Quanquan Gu. On the convergence of adaptive gradient methods for nonconvex optimization. *arXiv preprint arXiv:1808.05671*, 2018.

[251] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*, 2019.

[252] Yi Zhou and Yingbin Liang. Critical points of neural networks: Analytical forms and landscape properties. *arXiv preprint arXiv:1710.11205*, 2017.

[253] Yi Zhou and Yingbin Liang. Critical points of linear neural networks: Analytical forms and landscape properties. 2018.

[254] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[255] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep relu networks. *arXiv preprint arXiv:1811.08888*, 2018.

[256] Fangyu Zou and Li Shen. On the convergence of adagrad with momentum for training deep neural networks. *arXiv preprint arXiv:1808.03408*, 2018.

[257] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. *arXiv preprint arXiv:1811.09358*, 2018.